User's Guide

KLOCwork MSC to SDL Synthesizer

Cookbook of MSC Specifications

Release 2.0

Printed on 13 February, 2002

KLOCwork Corporation
Toll-free telephone: 1-866-556-2967
E-mail:
sales@klocwork.com
support@klocwork.com
Website: http://www.klocwork.com

Offices:
1 Antares Drive, Suite 510
Nepean, Ontario
Canada
K2E 8C4
(613) 224-2277

1700 Montgomery Street
Suite 111
San Francisco, CA
USA
94111
(415) 954-7154

Contents

Introduction

This volume, KLOCwork MSC to SDL Synthesizer *CookBook of MSC specifications*, contains a selection of examples that we hope will assist you to better understand the key concepts of the KLOCwork MSC for SDL Synthesizer tool. MSC to SDL Synthesizer analyzes scenario models specified as a set of Message Sequence Charts (MSC) and High-level Message Sequence Charts (HMSC) and automatically synthesizes executable state machine specifications in Specification and Description Language (SDL).

This document contains the following parts:

- Introduction. The rest of this section describes several ways in which the MSC to SDL Synthesizer can bring value to your organization.
- Using MSC to SDL Synthesizer stand-alone
- Using MSC to SDL Synthesizer with Telelogic Tau

Please refer to the Reference Manual of the KLOCwork MSC to SDL Synthesizer for systematic coverage of the installation, operations, and language-related issues of the tool. Please refer to the Tutorial on the KLOCwork MSC to SDL Synthesizer for best practices of using the MSC to SDL Synthesizer tool.

CHAPTER 1

Toll Terminal

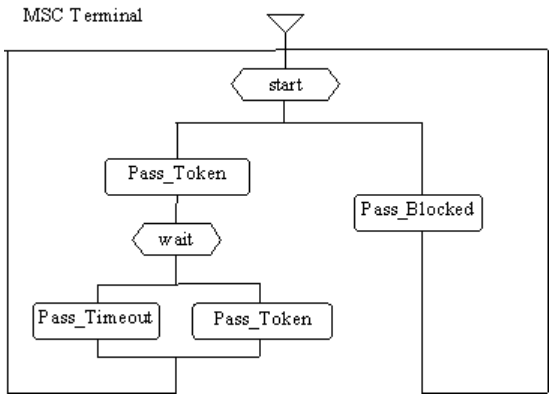
The example model describes a toll terminal (a system which collect fees for providing access to using a facility, e.g. a subway). The toll terminal is designed to grant access into a controlled zone only after a special token is dropped into a token-receiver. The toll terminal has a color access indicator. The red color means that access is forbidden. The green color means that access is granted. The terminal has also a sensor to detect whether the passenger has started to go through.

The toll terminal is the system under development in this MSC specification. The name of the system is 'Terminal'. A passenger going through the toll terminal is represented on MSC diagrams by an instance named 'Passenger'. The single scenario named 'Pass' describes a passage of the passenger through the terminal. There are one normal and two abnormal variants in this scenario:

1. The passenger drops a token and successfully passes through the toll terminal after it indicates that access is granted. This is a normal variant, that is describes sequential composition of MSCs Pass_Token and Pass_Enter.
2. The passenger drops a token but fails to enter through the toll terminal. If nobody goes through during a 60-second time interval the terminal rings and returns to its initial state. This means that the passage becomes again forbidden and the next token is expected. This is described by sequential composition of MSCs Pass_Token and Pass_Timeout.
3. The passenger tries to pass through the terminal without dropping a token. In this case the terminal blocks the passage. This is described by MSC Pass_Blocked.

MSC Pass_Token is the common part of scenarios 1 and 2. It ends with global condition 'wait'. MSCs Pass_Enter and Pass_Timeout start with the same global condition. MSC Pass_Token may be therefore composed with any of MSCs Pass_Enter and Pass_Timeout depending on which of scenarios 1 and 2 takes place. The behavior specified by this model is represented by HMSC 'Terminal'.

Figure 1: Representation of model Toll Terminal on HMSC Terminal



All source files in MSC PR format and Telelogic Tau diagrams and synthesized SDL executable model can be found in MSC to SDL installation directory in doc/examples/Terminal.

Figure 2: MSC Pass_Enter

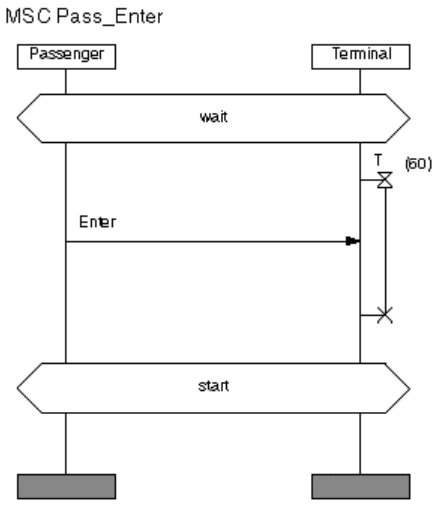


Figure 3: MSC Pass_Token

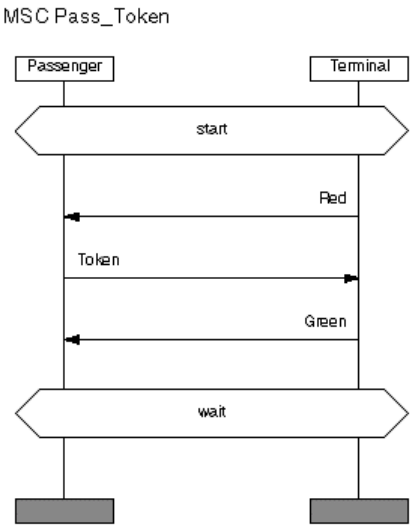
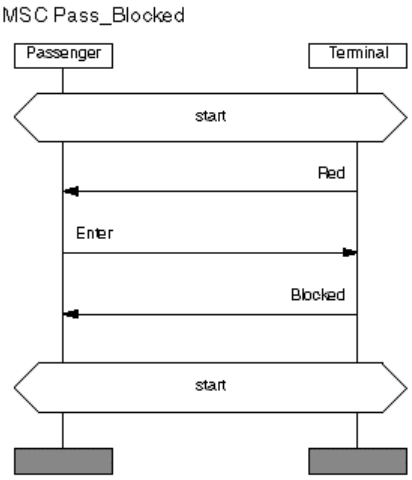


Figure 4: MSC Pass_Blocked



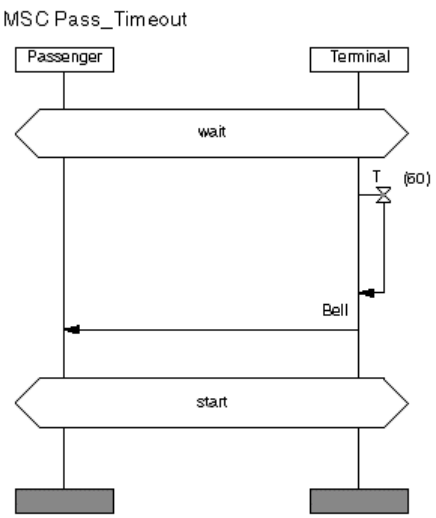


Figure 5: MSC Pass_Timeout

CHAPTER 2

Counter

The example describes a simple counter with the following capabilities:

- The counter provides capability to reset its value to zero;
- The counter has capability to increment its value by one;
- The counter has capability to display its current value.

The counter is the system under development in the MSC specification. The name of the system is 'Counter'. External entity accessing the variable is named 'User'.

The following three scenarios are present in the specification:

Scenario	Description	MSC diagram	Interaction
Reset	set the value of the counter to zero	Counter_Reset	The user sends a message 'S_Reset' to the counter
Incr	increment the value of the counter by one	Counter_Incr	The user sends a message 'Incr' to the counter
Display	obtain the value of the counter	Counter_Display	The user sends a message 'Display' to the counter. The counter returns its value with a message 'Show'.

MSC Counter_Init specifies the initialization of the counter. At the condition 'a_loop' the choice between three use cases is made. The behavior specified by this model is represented by HMSC 'Counter'.

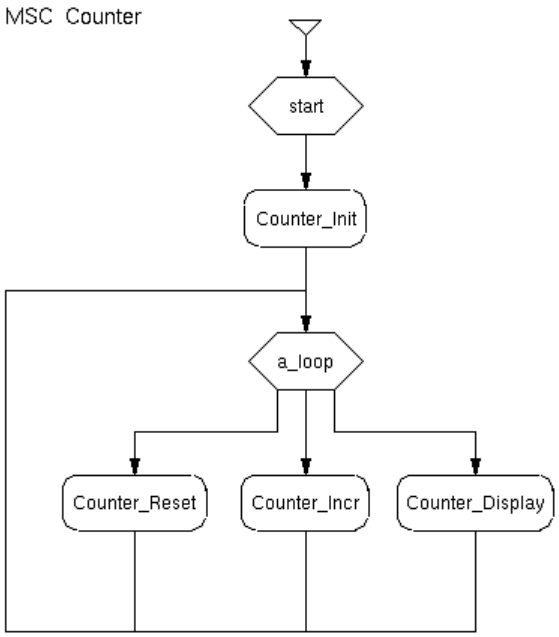


Figure 6: Model "Counter" in HMSC representation

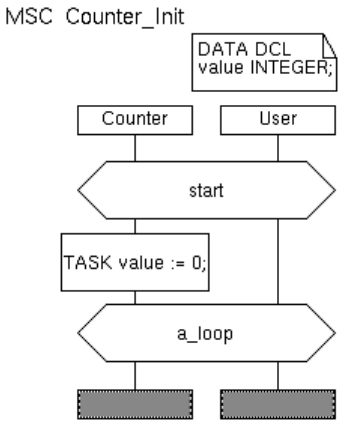


Figure 7: Initialization of the counter

Figure 8: Display of the current value of the counter

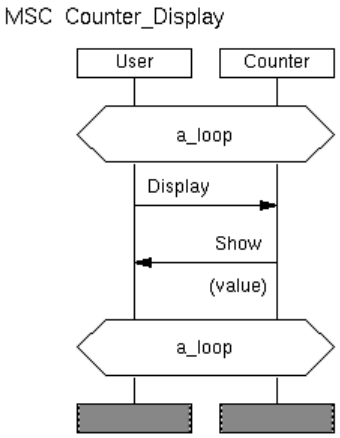


Figure 9: Incrementing the counter

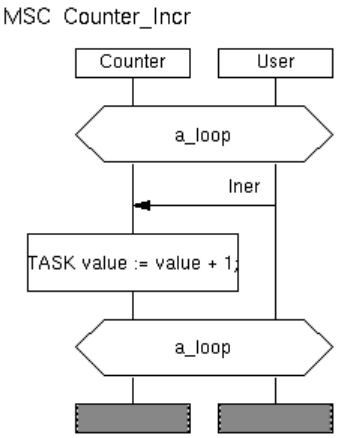
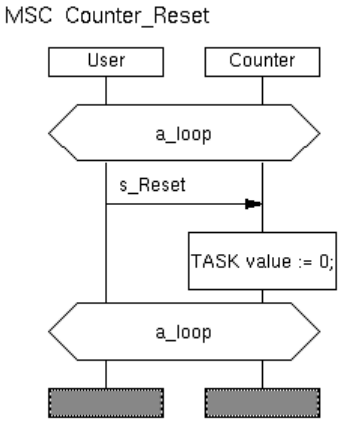


Figure 10: Resetting the counter



CHAPTER 3

Calculator

The example describes a simple calculator, which is capable of computing sums of integers. The calculator has three buttons, numeric pad and a digital display. The names of the buttons are 'Clear', 'Add' and 'Done'. The 'Init' button is used to start computing a sum, the numeric pad is used to input summands, the 'Add' button is used to enter each next summand and add it to the sum, and the 'Done' button is used to get the computed sum on the display.

The calculator is the system under development in this MSC specification. The name of the system is 'Calculator'. On MSC diagrams, instance with name ‘User’ represents the user of the calculator. The single scenario named 'Sum' describes a computation of a sum. Incorrect actions of the user and abnormal situations are not considered. The computation of each sum is described by MSCs in the following way:

- Pressing the 'Clear' button at the beginning of computation is represented on MSC Sum_Init as sending an 'Init' message from instance 'User' to instance 'Calculator'.
- An input of each next summand on a numeric pad and the following pressing the 'Add' button is represented on MSC Sum_Add as sending an 'Add' message. The parameter of this message is the number been input.
- The pressing of the 'Done' button by the user is represented on MSC Sum_Display as sending a 'Done' message. The calculator replies with a 'Display' message with the computed sum as a parameter. After this a computation of a new sum can start.

The behavior specified by this MSC specification is represented by HMSC 'Calculator'.

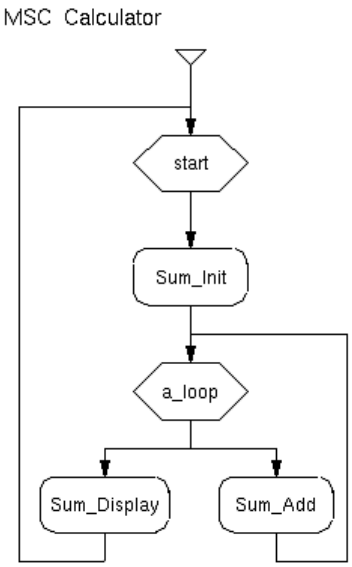


Figure 11: Model "Calculator" in the HMSC representation

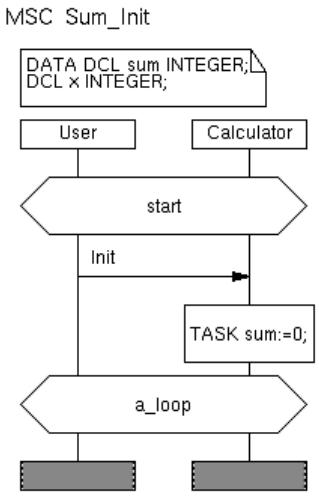


Figure 12: Initialization of the calculator

Figure 13: Displaying the result of the the calculation

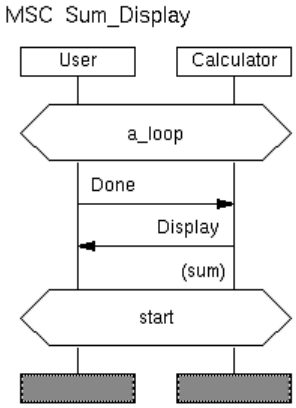
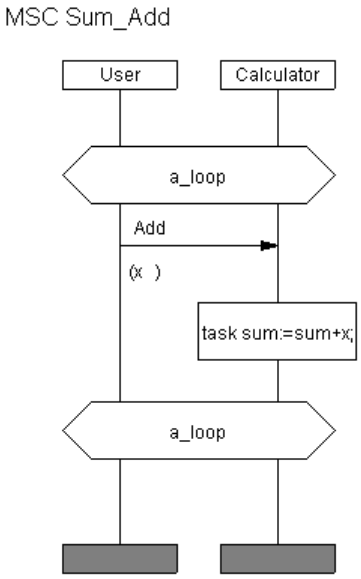


Figure 14: Add operation in the calculator



CHAPTER 4

Split

This example describes a pipe, which transforms a character-oriented data stream into a line-oriented data stream. The length of line is assumed to be 80 characters. The transformation inserts a line-feed character is inserted after every 80 transmitted characters. The pipe is the system under development in this MSC specification. The name of the system is 'Split'. External entities are the following:

- 'Sender' instance on MSCs represents the source of data, and
- 'Receiver' instance on MSCs represents the destination of data.

The single scenario named 'Transfer' describes transmission of a sequence of characters. First, the sender initializes the connection as described in MSC Transfer_Init. Then the characters are transmitted one by one. Transmission of each character can progress according to one of two scenarios:

1. If the character fits into the current line without exceeding its length it is simply passed to the receiver. This is described by sequental composition of MSCs Transfer_Read and Transfer_Char.
2. If the character doesn't fit into the current line it becomes the first character in the next line. In this case an additional line-feed character is passed to the receiver before it. This is described by sequental composition of MSCs Transfer_Read and Transfer_Ins
3. Finally, the transmission is terminated as shown in MSC Transfer_Done.

MSC Transfer_Read is the common part of scenarios 1 and 2. It ends with global condition 'send' where the choice between these two scenarios is made. This example uses variable declarations, message parameters and check conditions. An integer variable 'count' holds the number of characters in the current line. The choice between scenarios 1 and 2 is made depending on the value of this variable. Corresponding check conditions are located ot MSCs Transfer_Char and Transfer_Ins.

All source MSC PR files, Telelogic Tau diagrams and synthesized executable SDL model can be found in MSC to SDL installation directory in doc/examples/Split.

The behavior specified by this use case model may be represented by HMSC 'Split'.

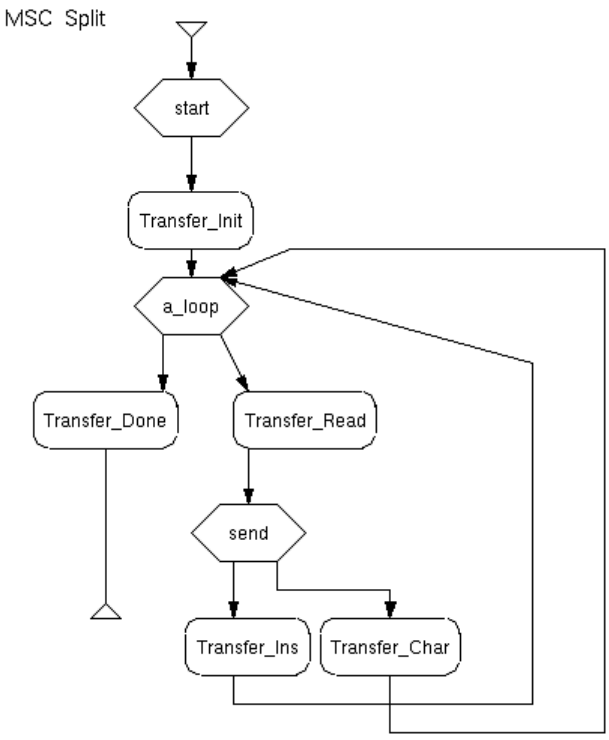


Figure 15: HMSC for the model Split

Figure 16: Transferring a single character

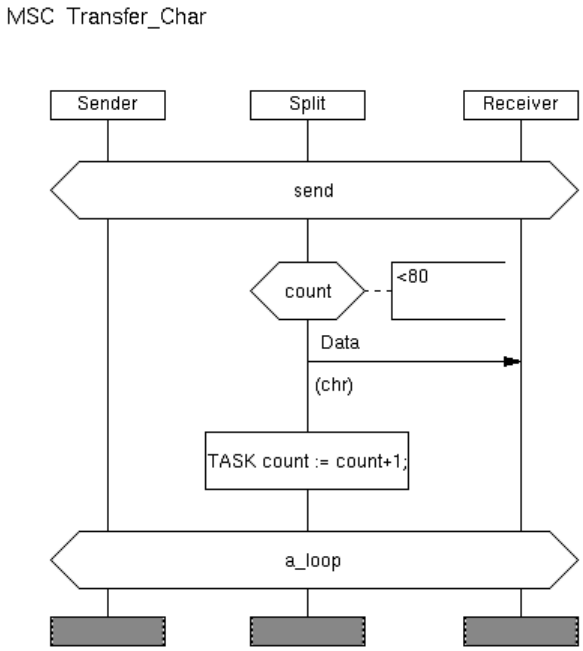


Figure 17: Initializing transfers

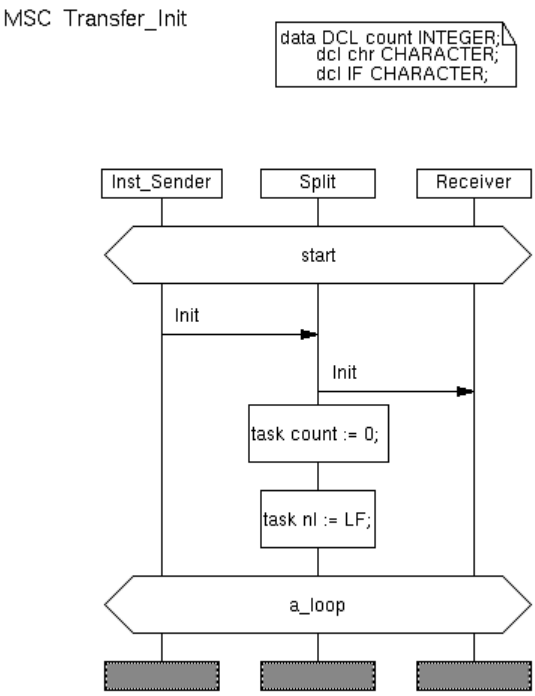


Figure 18: Inserting a line terminator

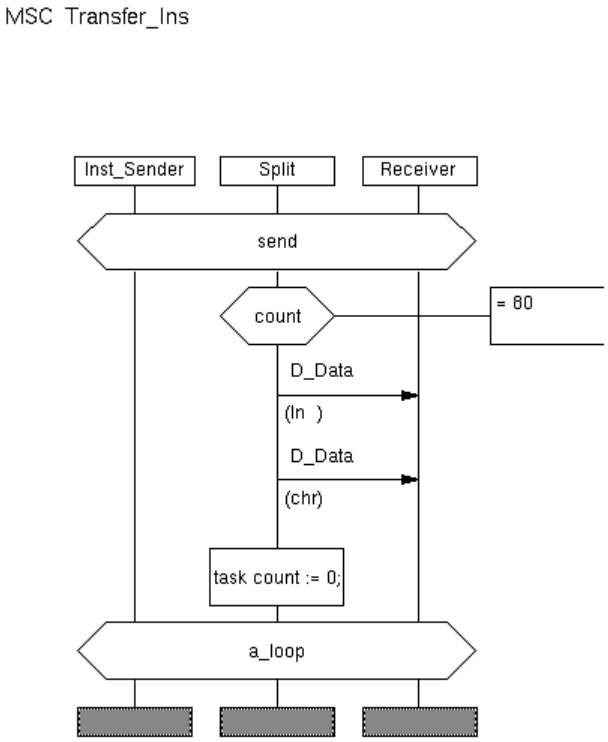


Figure 19: Some deterministic example for Sender

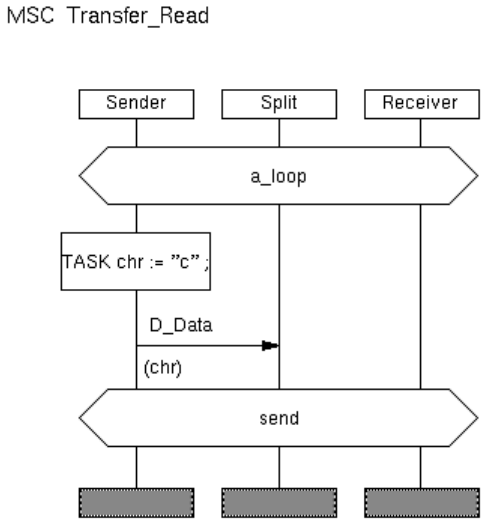
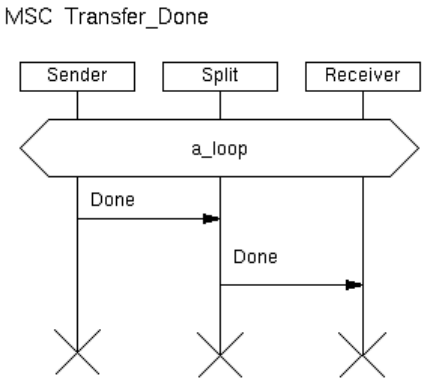


Figure 20: Terminating transfers



CHAPTER 5

Filter

This example describes a data processing unit (a filter), which gets a sequence of integer values and outputs only positive values preserving their order. This model is similar to the MSC specification in 'Split' example. The data processing unit is the system under development in the specification. The name of the system is 'Filter'. The external entities are the following:

- 'Sender' instance on MSCs represents the source of data, and
- 'Receiver' instance on MSCs represents the destination of data.

The single scenario named 'Process' describes processing of a sequence of values. First, the sender initializes the data processing session as described in MSC Process_Init. Then the values are processed one by one. Processing of each value can progress according to two scenarios:

1. If the value is positive then it is simply passed to the receiver. This is described by sequental composition of MSCs Process_Generate and Process_Send.
2. If the value is negative or zero it is skipped without passing it to the receiver. This is described by sequental composition of MSCs Process_Generate and Process_Skip.

Finally, the processing is terminated as shown on MSC Process_Done. After that the new data processing session may be initiated. Note that this is unlike 'Split' example where only one transmission session is allowed.

MSC Process_Generate is the common part of scenarios 1 and 2. It describes the generation of a random value in the 'Sender' actor. It ends with a global condition 'process' where choice between these two scenarios is made. The choice is done based on the sign of the value. Corresponding check conditions are located in MSCs Process_Send and Process_Skip for this purpose. The behavior specified by this model is represented by HMSC 'Filter'.

All source MSC PR files, Telelogic Tau diagrams and synthesized executable SDL model can be found in MSC to SDL installation directory in doc/examples/Filter.

Figure 21: HMSC for model Filter

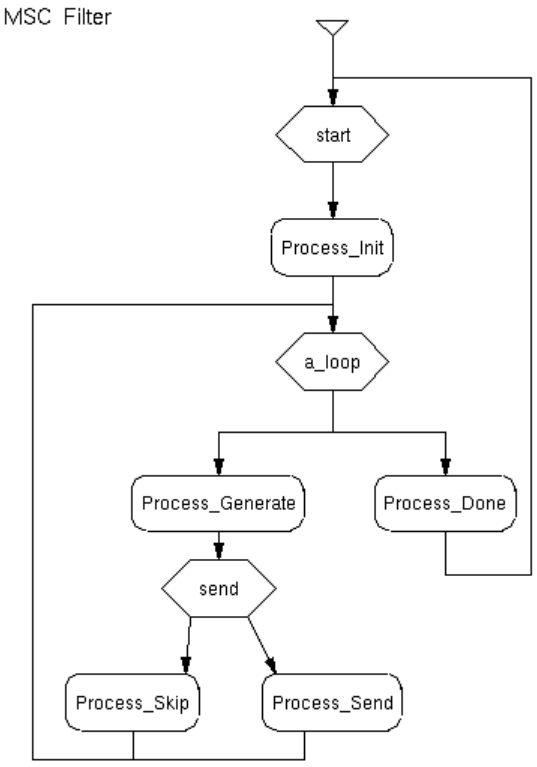


Figure 22: Random generation of values by Sender

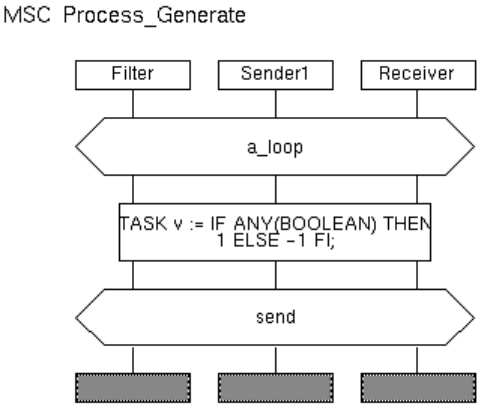
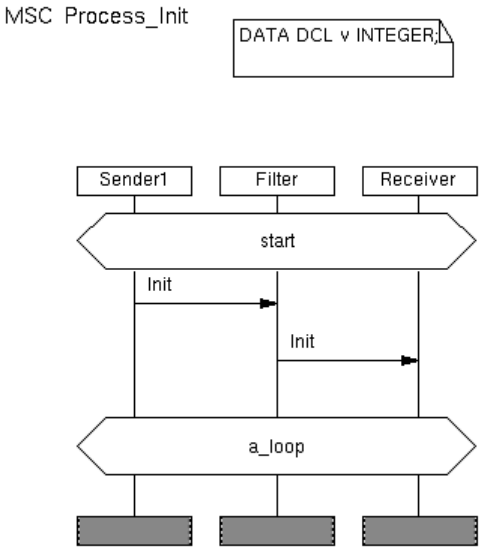


Figure 23: Initializing the Filter



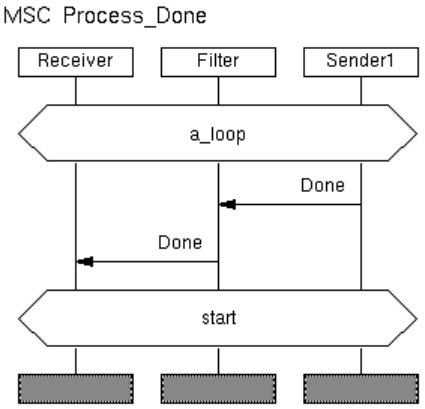


Figure 24: Terminating the Filter

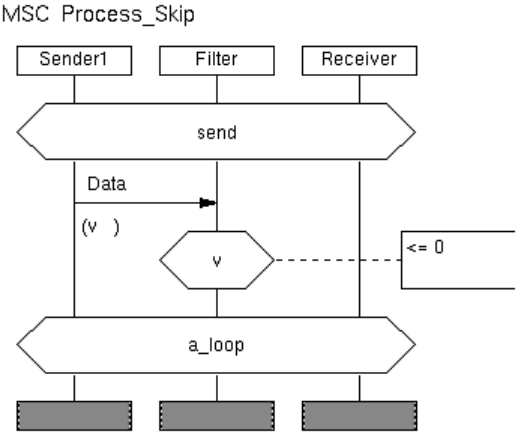
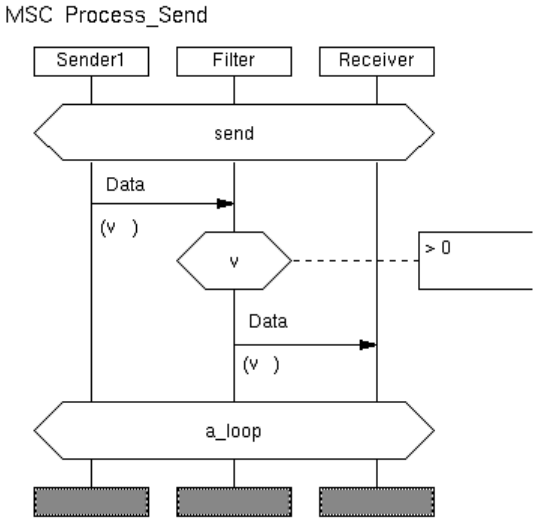


Figure 25: Skipping over negative inputs

Figure 26: Transferring positive values



CHAPTER 6

Three-tier system

This example describes a simplified three-tier system, which consists of a Client component, a Server component and a Repository component. The Client interacts with the Server by sending requests for computations, data and status. The Server handles requests from the Client and uses the Repository component to handle all the data aspects (performing computations and storing results). The Server component implements the business logic of the system.

In This Chapter

Components and responsibilities

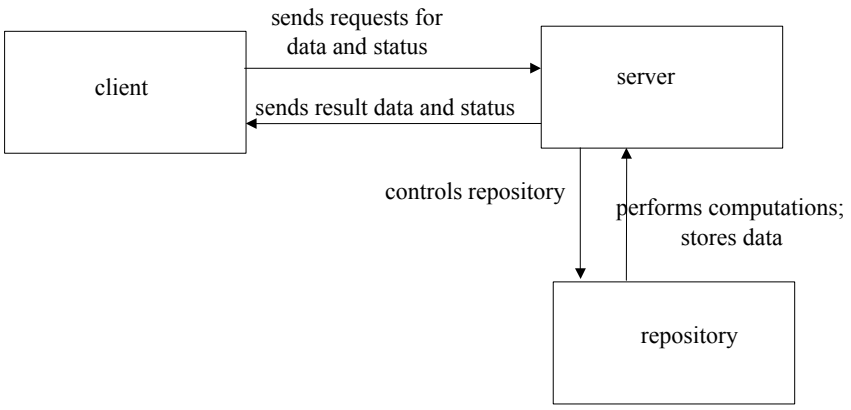
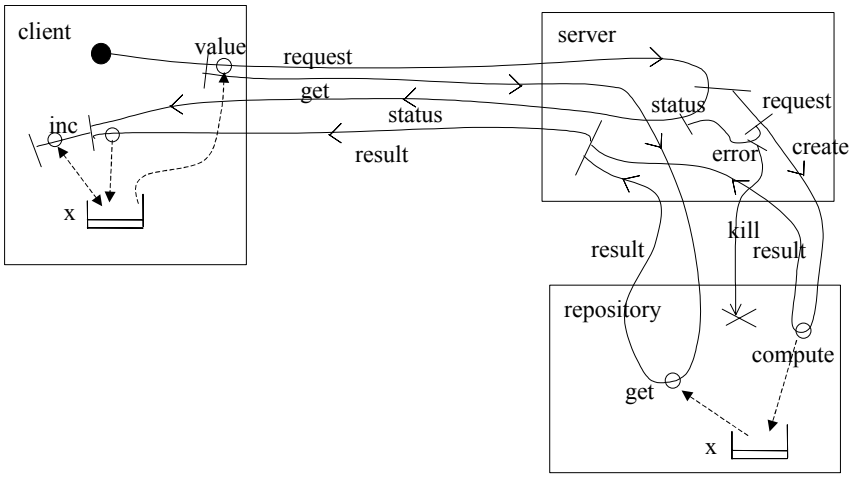


Figure 27: Roles and responsibilities of our simple 3-tier system

Figure 28: Use case map of our simple 3-tier system



MSCs

Figure 29: Client requests computations from the Server

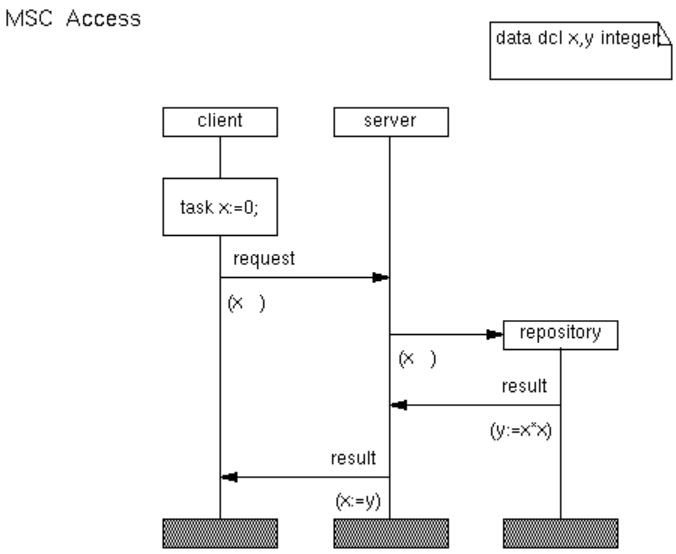


Figure 30: Use case map for MSC access

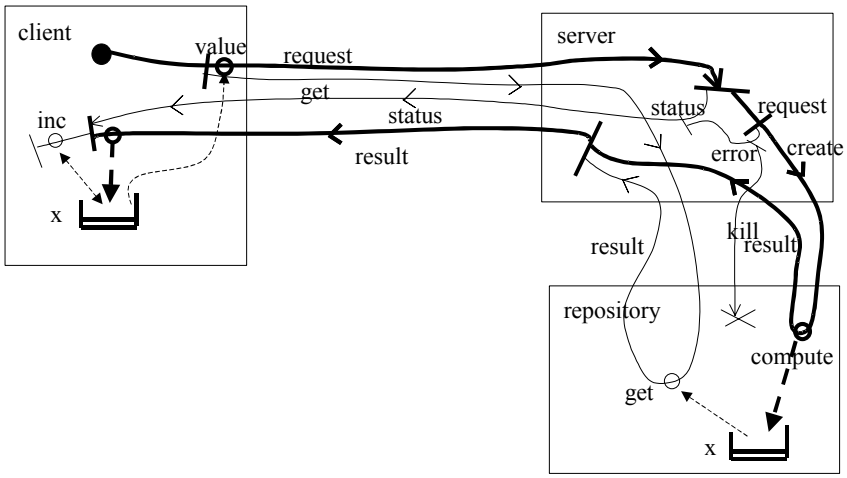


Figure 31: Client requests data from the Server

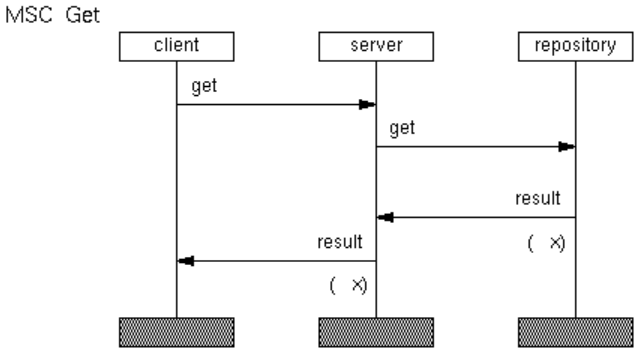


Figure 32: Use case map for MSC get

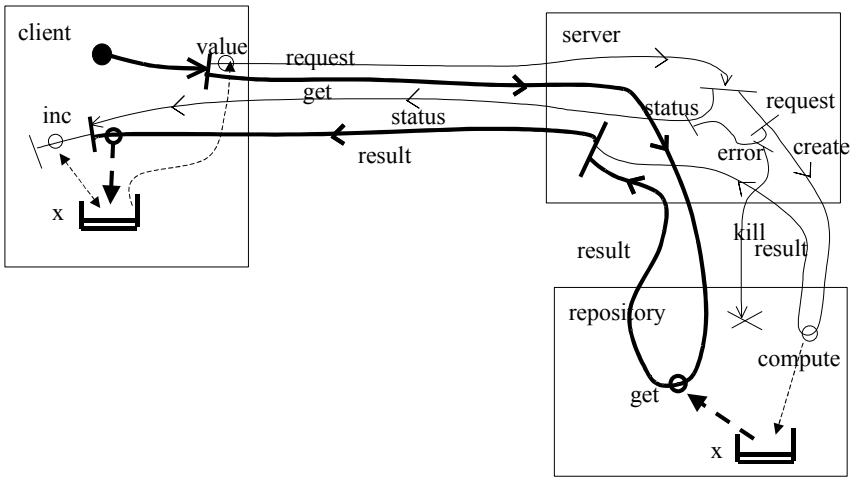


Figure 33: Client requests status from Server

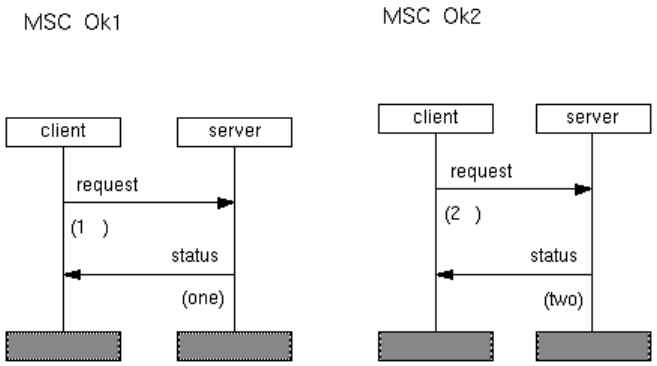


Figure 34: Use case map for MSCs ok1, ok2

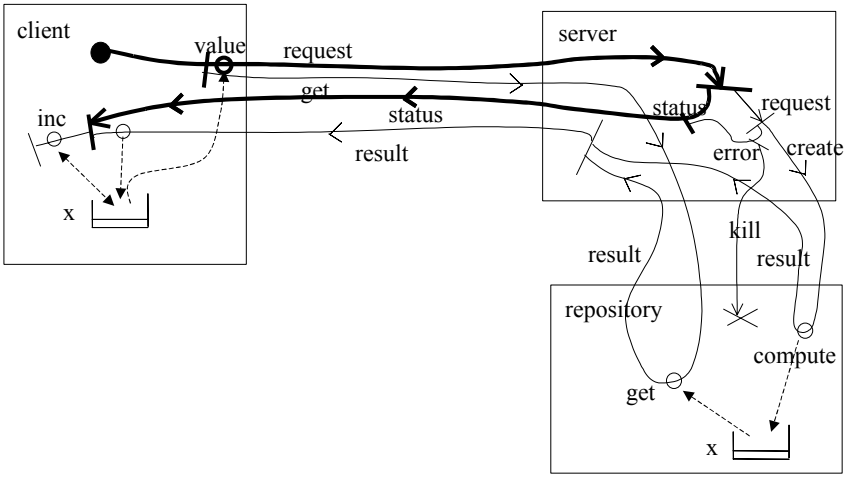


Figure 35: Server informs Client about failed requests

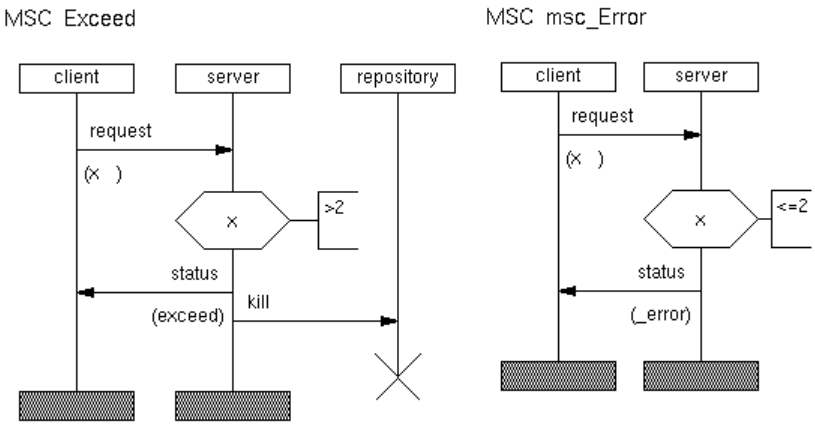


Figure 36: Use case map for MSC exceed

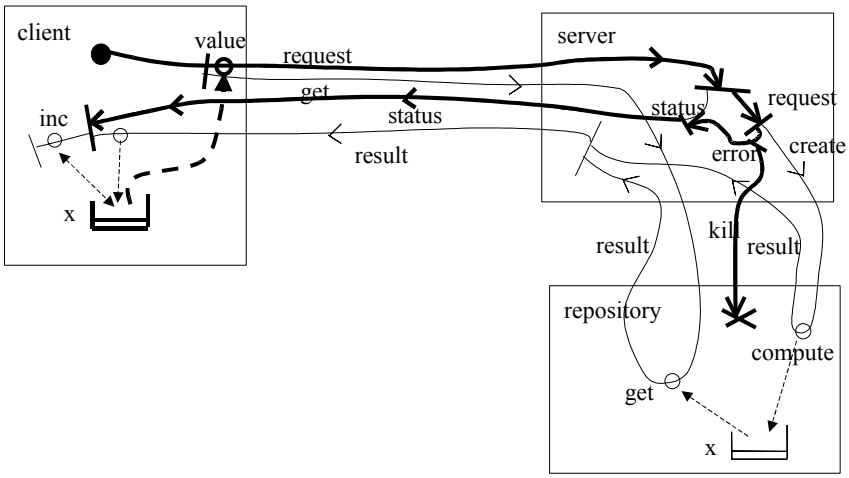


Figure 37: Use case map for MSC msc_error

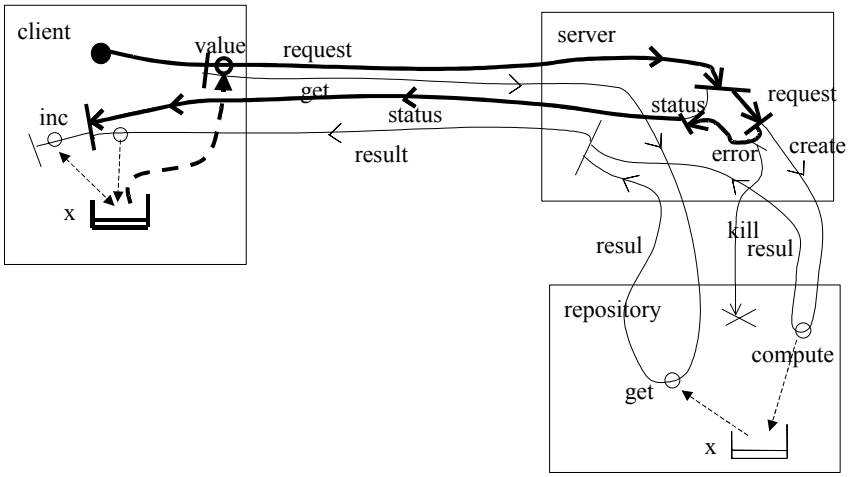


Figure 38: Client-side computations

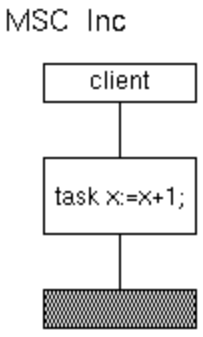
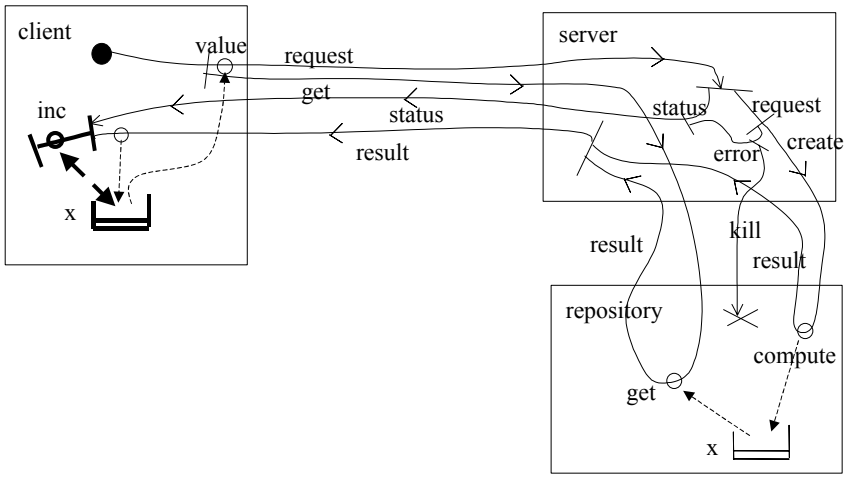
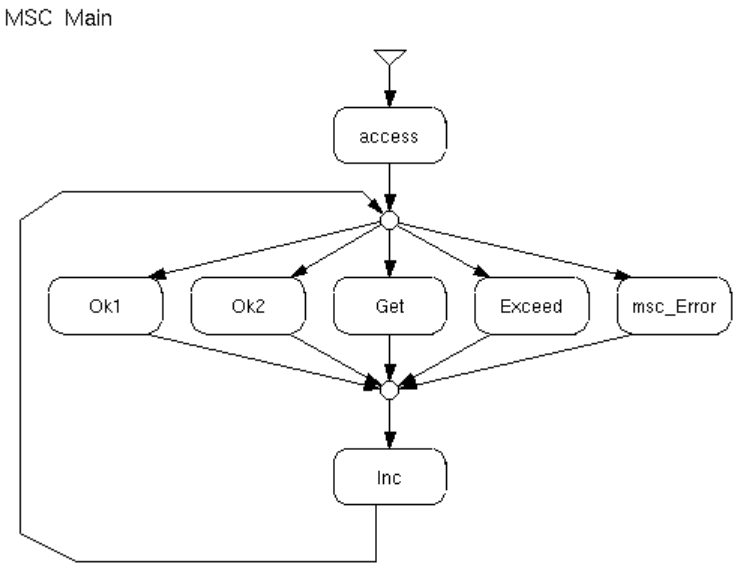


Figure 39: Use case map for MSC inc



HMSC

Figure 40: HMSC for our 3-tier system



Synthesized SDL model: Structure

Figure 41: SDL system for our 3-tier system

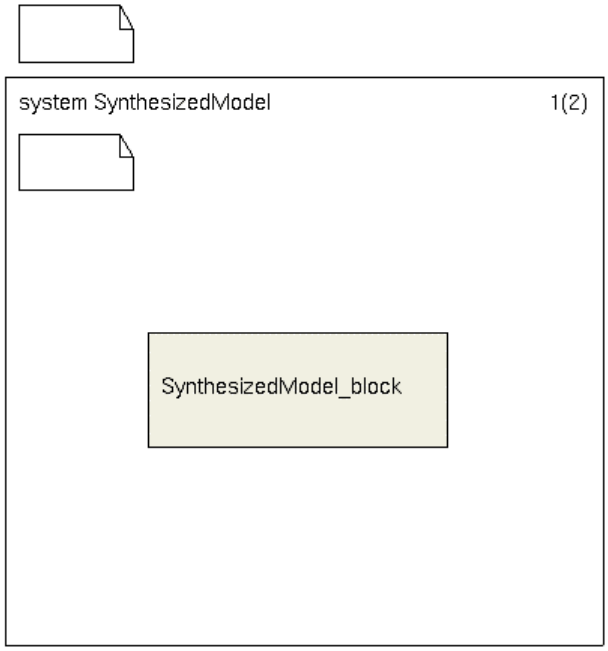


Figure 42: Synthesized architecture of the 3-tier system

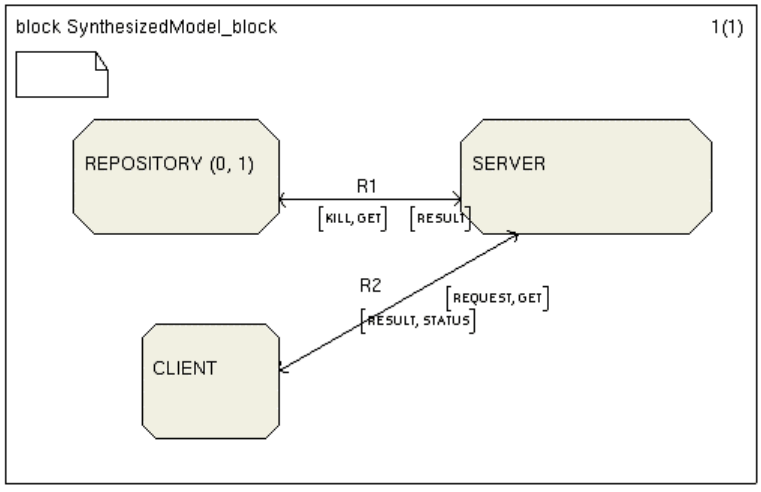
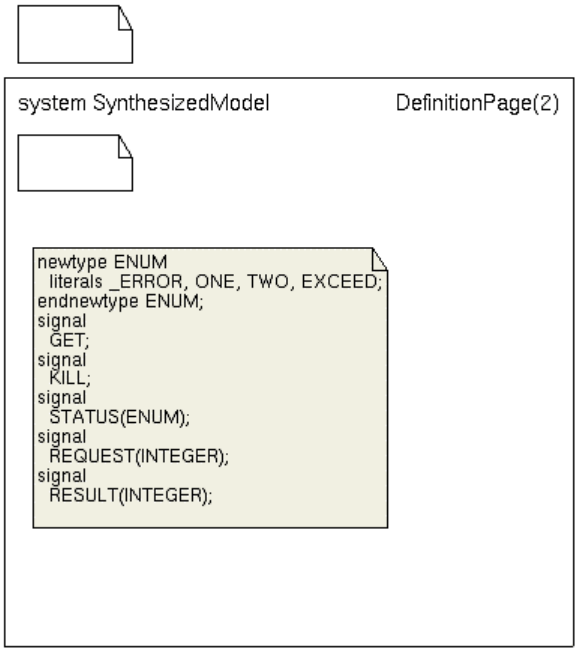


Figure 43: Synthesized global data definitions for our 3-tier system



Synthesized SDL Model: Client component

Figure 44: Use case map for Client component

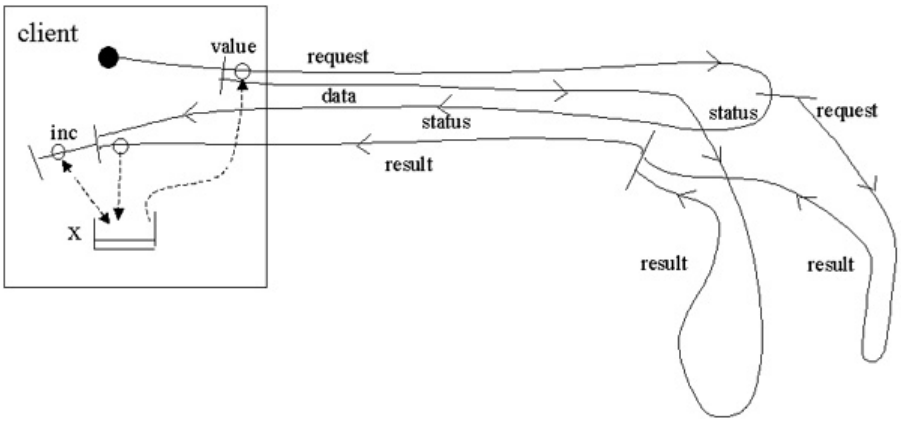


Figure 45: Synthesized data definitions for the Client component

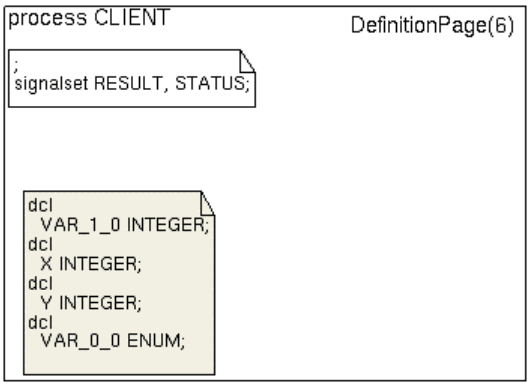


Figure 46: Initialization and the first computation request in Client

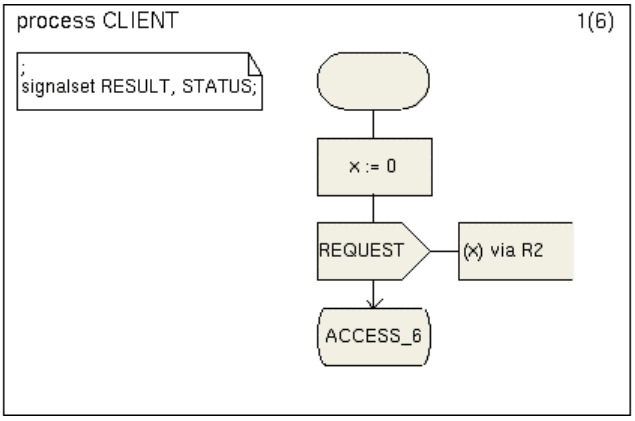
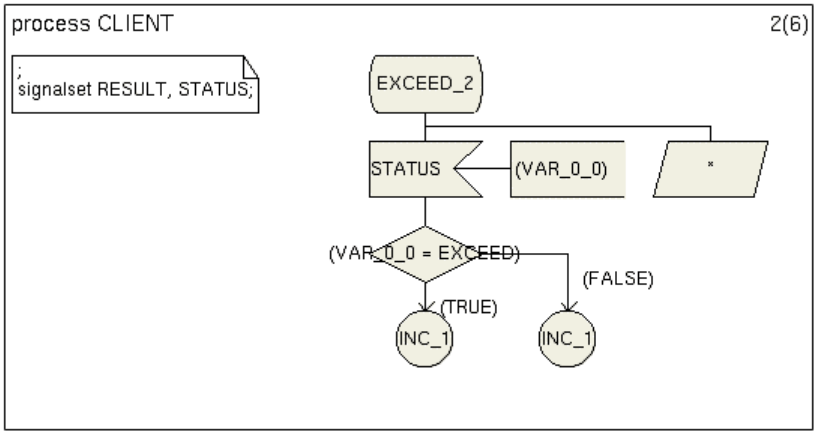


Figure 47: Processing results of computation request



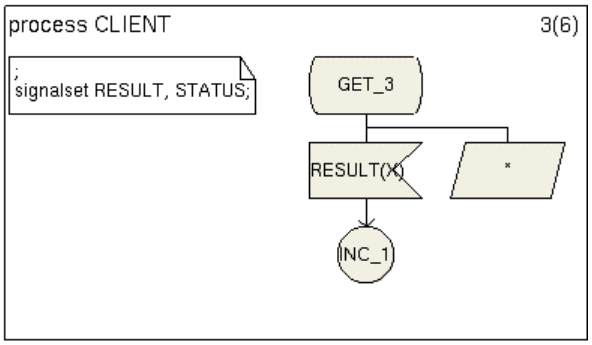


Figure 48: Processing results of the data request

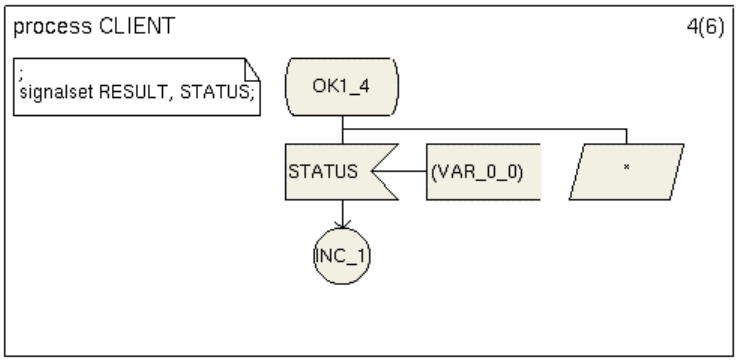


Figure 49: Processing results of the status request

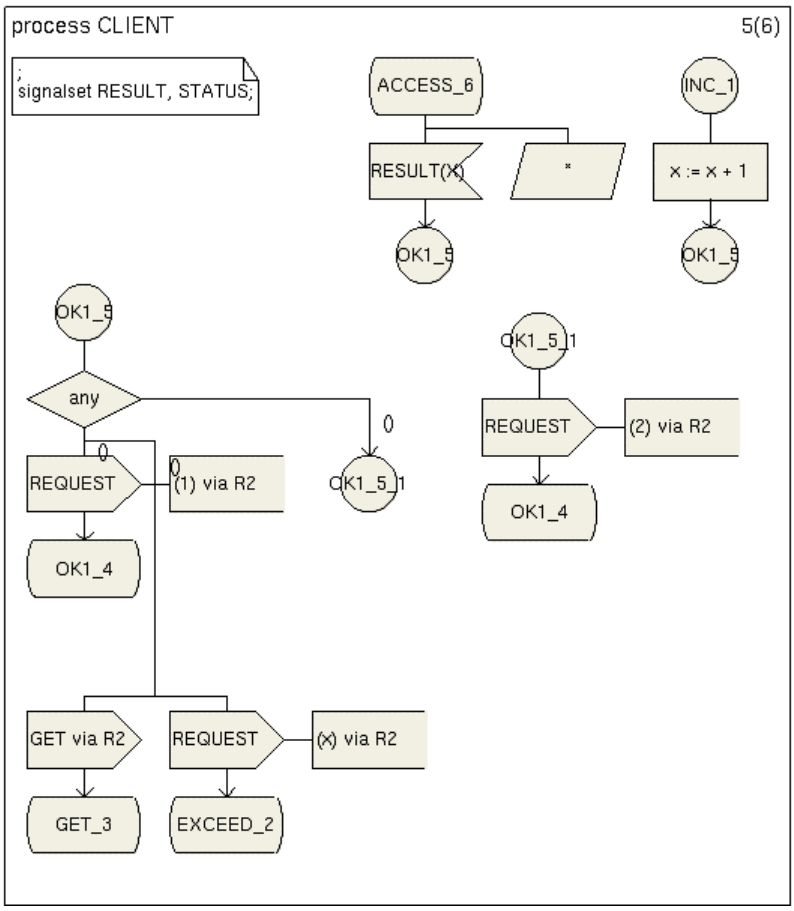


Figure 50: Logic of the Client component

Synthesized SDL model: Server component

Figure 51: Use case map for the Server component

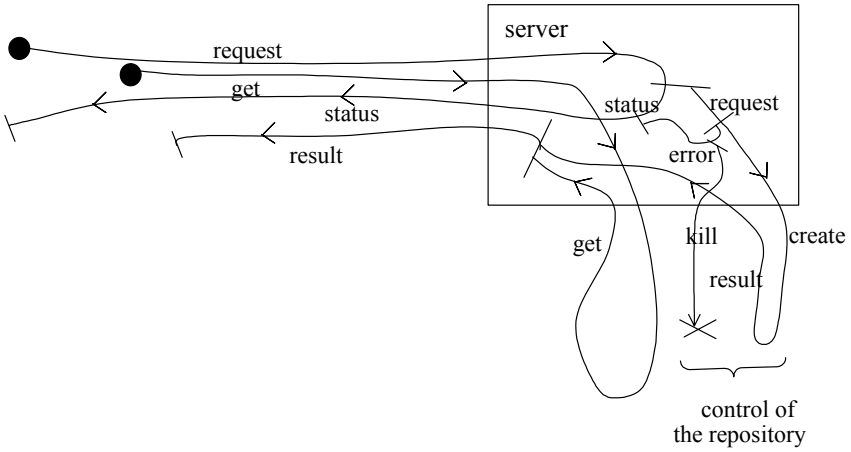


Figure 52: Synthesized data definitions for the Server component

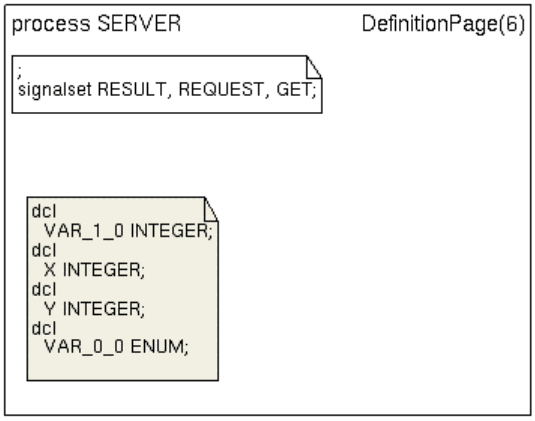


Figure 53: Initialization of the Server component

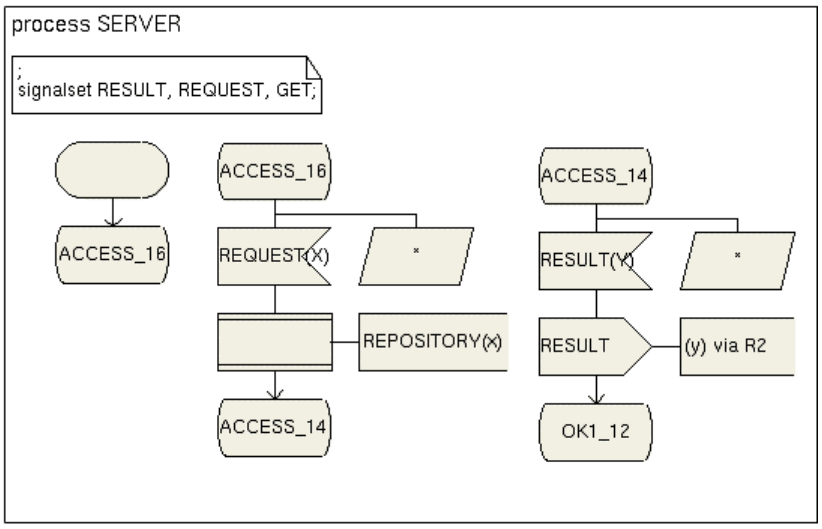


Figure 54: Logic of the Server component

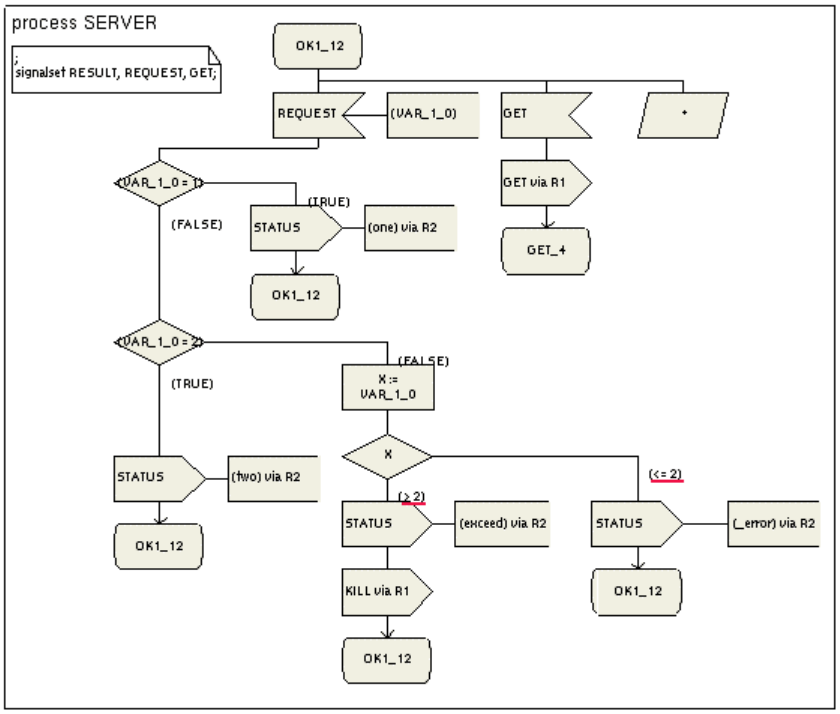
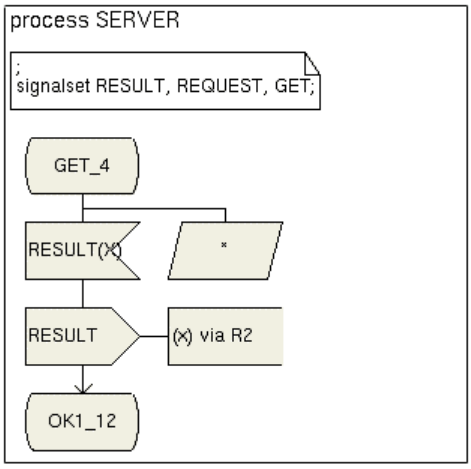


Figure 55: Handling communication with the Repository



Synthesized SDL model: Repository component

Figure 56: Use case map for the Repository component

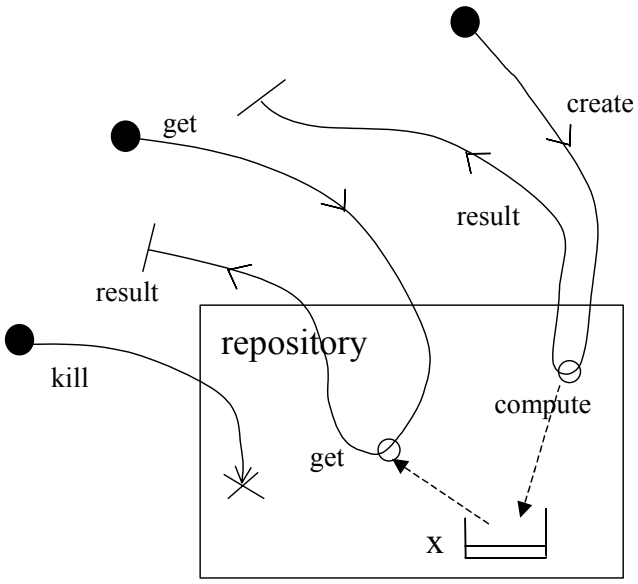


Figure 57: Synthesized data definitions for the Repository component

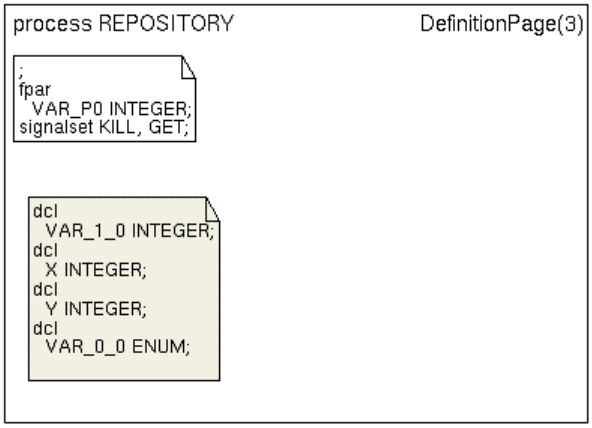
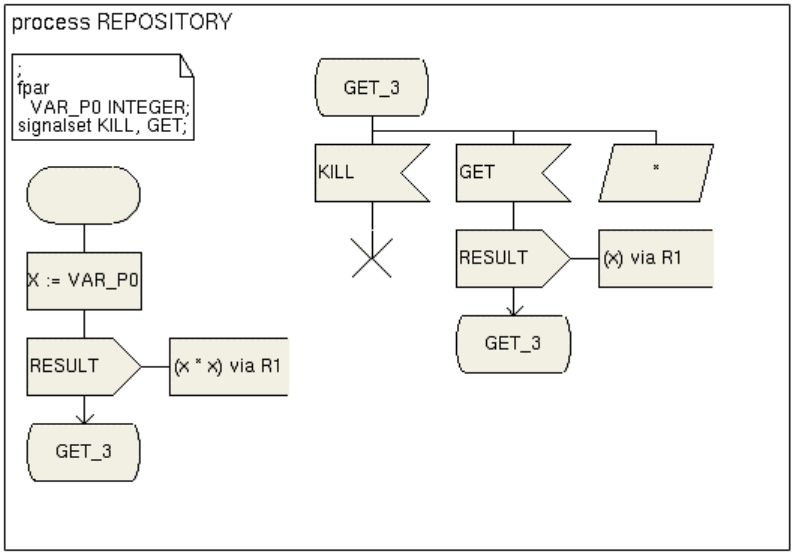


Figure 58: Logic of the Repository component



Index