

EECS 223:

Introduction to Real-Time Distributed Programming

Lecture : Remote service calls
to TMOs containing SvMs only



Table of contents

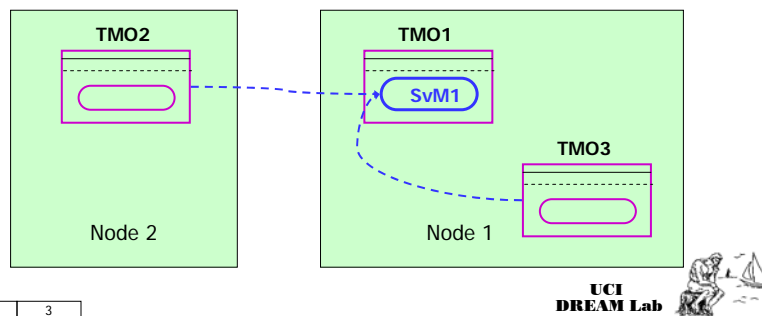
- Introduction
 - Time-triggered Message-triggered Object (TMO) Structuring Scheme
 - Interaction among TMO's
 - Return Deadline vs. Guaranteed Service Time
 - SvM Gate and location transparency
- Creation of an SvM (Service Method)
 - Definition of SvM
 - An example of an SvM Method
- SvM Gate
 - Definition of SvMGateClass
 - Examples of programs using SvMGateClass
- Program Examples



SvM (Service Method)

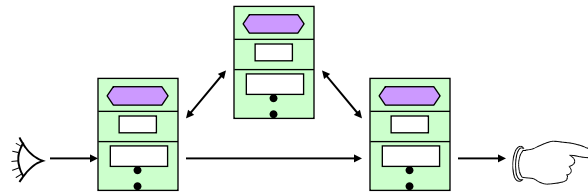
- A slight (?) extension of the member function in a C++/Java object
- The extension is in allowing local clients (objects hosted on the same node) and remote clients to call a particular SvM exactly in the same manner (i.e., no more or less hassles).

=> TMO is a distributed computing extension of the C++/Java object.



Apr-05 3

Interaction among TMO's

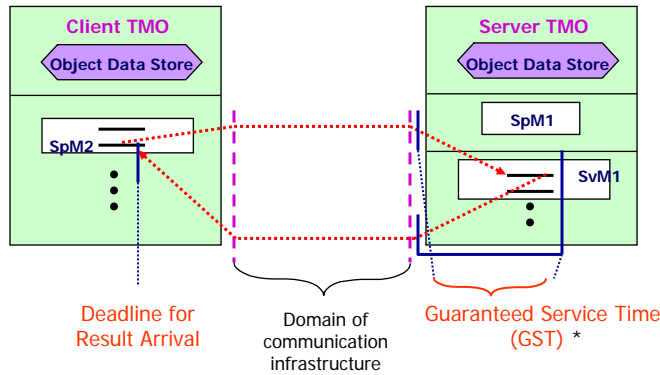


- Mode 1: via Remote SvM calls
 - Blocking calls with return deadlines imposed
 - Non-blocking calls and subsequent result checks with deadlines imposed
 - Client-transfer calls
- Mode 2: via General message multicasts over RMMCs (Real-time Multicast and Memory-replication Channels)
 - Event messages
 - State messages

Apr-05 4

UCI DREAM Lab

Return Deadline vs. Guaranteed Service Time



* A violation of this deadline is a fault.

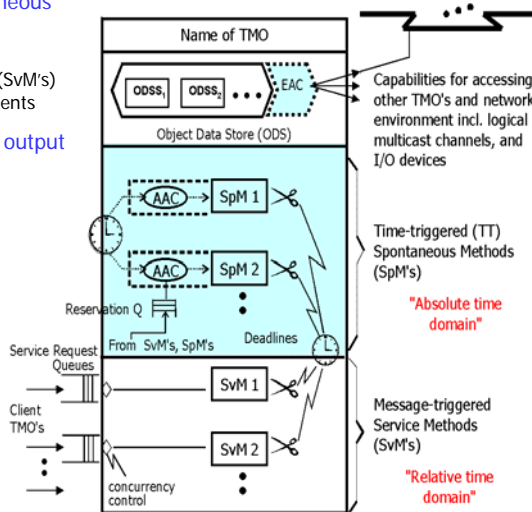
* If GST is violated, *Fault handling actions must take place.*

- * **Deadline for result arrival** - Call initiation time
 - > Max trans times imposed on communication infrastructure + GST
 - > Time consumed by communication infrastructure + GST

Apr-05 5

TMO (Time-triggered Message-triggered Object)

- **Time-triggered (TT-) or spontaneous methods (SpM's):**
 - Clearly separated from the conventional service methods (SvM's) triggered by messages from clients
- **Time-window** imposed on each **output action** and **method completion**
- **Connections to the network environment** as possible data members:
 - Real-time Multicast & Memory-replication Channels
 - **TMO access capabilities (possibly remote TMO's)**
- **Basic concurrency constraint (BCC):**
 - SpM executions not disturbed by SvM executions.
 - Eases design-time guarantee of timely services of TMO's



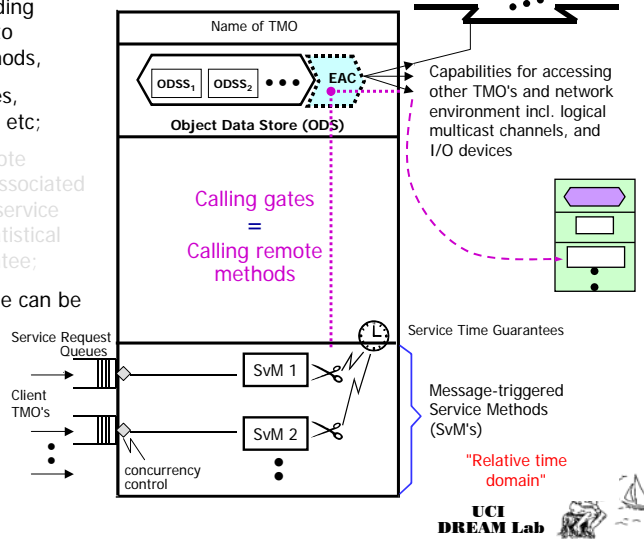
Apr-05 6

TMO

- High-level distributed computing component:

EAC (Environment Access Capability) section (an ODS extension) provides

- *Gate* objects providing efficient call-paths to remote object methods;
- I/O device interfaces, channel interfaces, etc;
- Each gate to a remote service method is associated with a guaranteed service time bound or a statistical service time guarantee;
- Client's call to a gate can be associated with **deadline for result return**, nothing more;



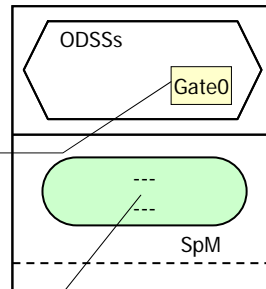
Apr-05 | 7

Ease of Creation of a Remote Service Call

- Just two statements are needed to create a remote service call to an SvM in another TMO !!

```
SvMGateClass Gate0 (_T("TMO2"),
    _T("SvM2"),
    tm4_DCS_age (7*1000*1000));
```

```
Gate0->NonBlockingSR (& SvM2Para,
    sizeof (SvM2Para), Timestamp);
```

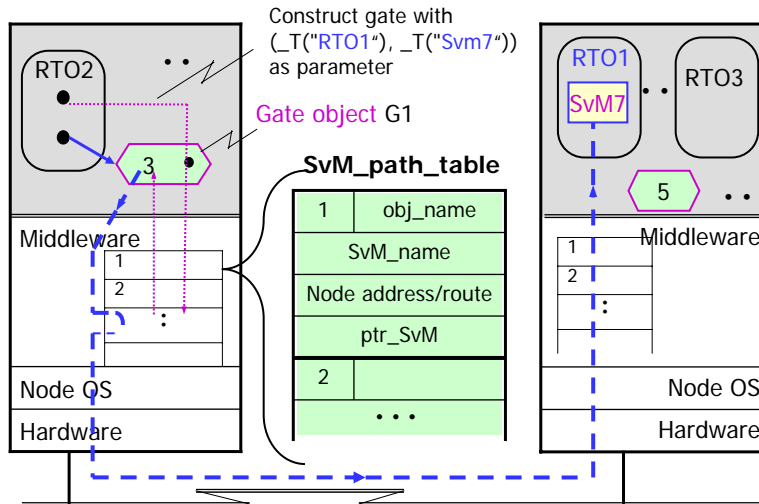


- The details will become clear later in this lecture.

Apr-05 | 8



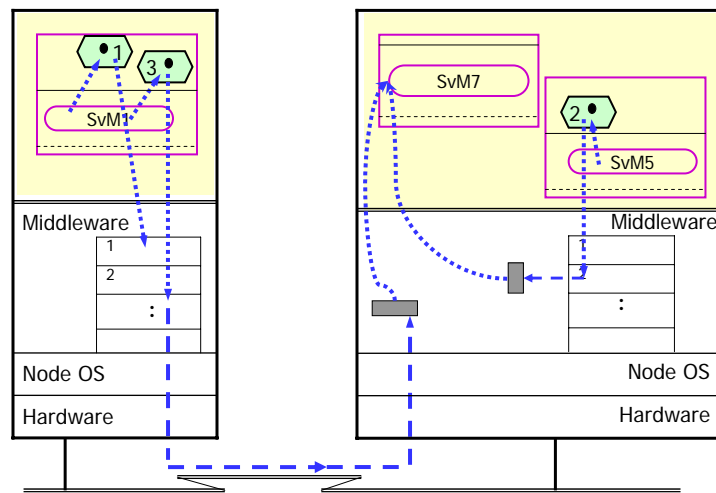
SvM Gate & Location Transparency



- A gate object is a front-end interface of an automatically created call-path to a remote SvM. Client objects can just use the gate's name in calling the associated remote SvM.

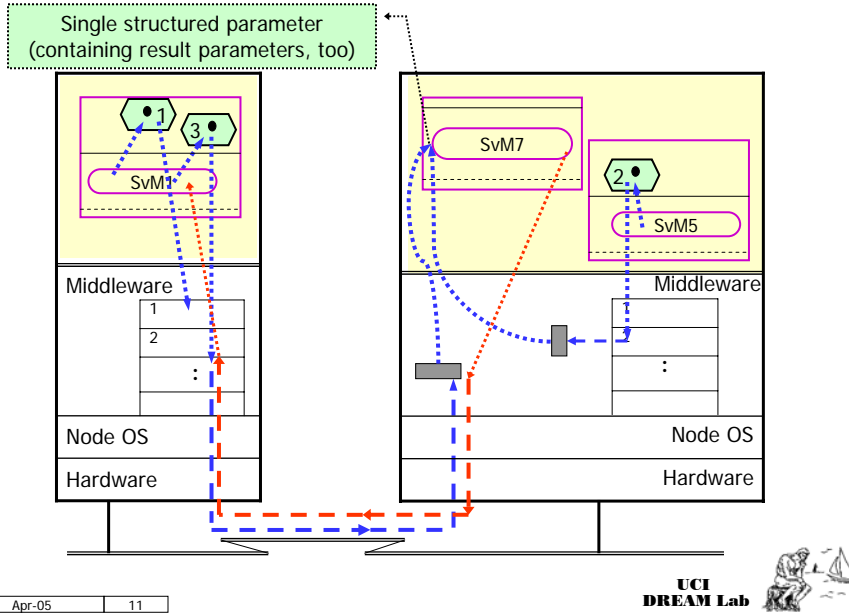
Apr-05 9

SvM Gate & Location Transparency (cont)

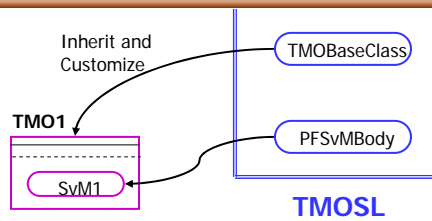


Apr-05 10

SvM Gate & Location Transparency (cont)

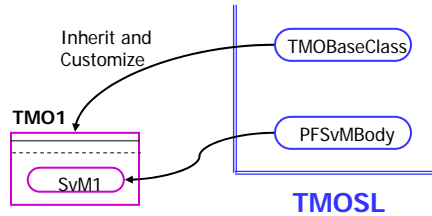


Creation of an SvM & an Enclosing TMO



- SvMs are declared as ordinary member functions of a TMO class.
- Each SvM should be registered to TMOSM with the following parameters:
 - Set of the **names and access modes of the ODSSs**
 - The **external name** which is a globally recognized symbolic name to the world outside the enclosing TMO

Creation of an SvM & an Enclosing TMO



- Guaranteed execution time bound (**GETB**)
- The maximum allowed number of concurrent executions of the same SvM (**PipelineDegree**)
- the maximum invocation rate (**MaxInvocations**) and/or the minimum invocation interval (**BasicPeriod**).

Apr-05 13



Definition of TMOBaseClass - Basic Data Types

```
enum access_mode_type {RO, RW}; // RO : Read Only,  
//RW : Read and Write
```

```
struct ODSS_n_access_mode  
{  
    int ODSS_ID;  
    access_mode_type access_mode;  
};
```

```
struct SvM_RegistParam  
{  
    TCHAR Name [MAX_SVM_NAME_SIZE];  
    MicroSec GETB;  
    int PipelineDegree;
```

Apr-05 14



Definition of TMOBase Class - Basic Data Types (Cont)

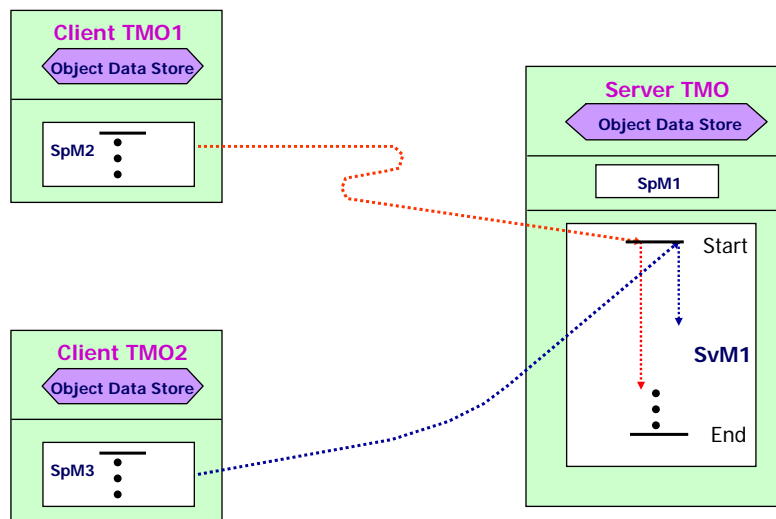
```

int      MaxInvocations;
        // Maximum number of invocations of subject SvM per
        // BasicPeriod that will be honored by the exec engine.
MicroSec BasicPeriod; // Default = 1000 * 1000
list <ODSS_n_access_mode*> ODSS_access_mode_list;
void build_regist_info_ODSS (int odss_id, access_mode_type mode);
~SvM_RegistParam ();
};
    
```

- 1) Since an SvM can access more than one ODSS, "ODSS_access_mode_list" should be a list of items, each defining the ODSS access mode for each ODSS.
- 2) "pipelining_degree" specifies the maximum allowed number of concurrent executions of the same SvM.

Apr-05 | 15

Def. of SvMBase Class - Basic Data Types (Cont) Pipelined Execution



Apr-05 | 16

Definition of TMOBase Class - Basic Data Types (Cont)

```
class CTMOBase
{
public:
    virtual ~CTMOBase ();

    // register an TMO to the middleware
    bool Register_TMO (TMO_RegistParam *);

    // When an SR is issued from the client side of application, a
    // timestamp for this SR is generated inside the exec engine and
    // sent to the server side.
    // This API can be called inside the SvM function body on the server
    // side to get the SR timestamp produced by the client side.
    MicroSec GetSRTimestamp ();
```

Apr-05

17

UCI
DREAM Lab



Definition of TMOBase Class - Basic Data Types (Cont)

```
protected:
    CTMOBase (void);

    // register an SpM/SvM to the middleware
    Bool Register_SpM (PFSpMBody, SpM_RegistParam *);

    Bool Register_SvM (PFSvMBody, SvM_RegistParam *);
};
```

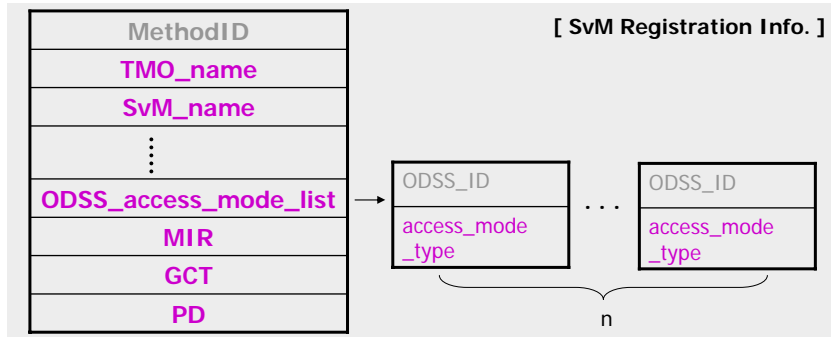
Apr-05

18

UCI
DREAM Lab



Definition of SvMBaseClass - Registration



Apr-05 | 19



Definition of SvMBase Class - Registration (Cont)

```

// An illustration of building SvM registration information.
TMO1::TMO1 (TCHAR* TMO_external_name,
           tms   TMO_start_time1)
{
    // register SvM
    SvM_RegistParam svm_spec;
    svm_spec.GETB = 20 * 1000;
    svm_spec.MaxInvocations = 100; // May be omitted.
    svm_spec.PipelineDegree = 3; // May be omitted.
    _tcscopy (svm_spec.Name, _T("SvM7"));
    svm_spec.build_regist_info_ODSS (ODSS1.GetId (), RO);
    RegisterSvM ((PFSvMBody) SvM1, & svm_spec);

    // register TMO
    ...}
    
```

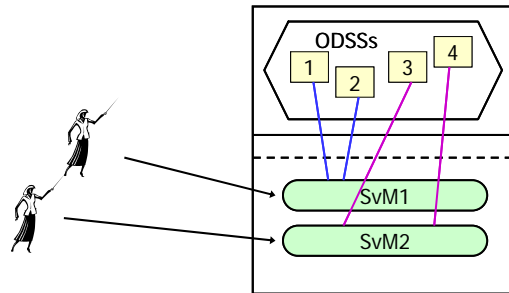
- This registration enables the execution engine to (1) receive SvM calls, (2) reserve ODSSs to be used in the SvM execution, (3) enforce a completion deadline, and (4) enforce concurrency limit.

Apr-05 | 20



Definition of SvMBase Class - Registration (Cont)

- Why register ODSSs ?
 - If TMOSM knows which ODSSs are used by each SvM, then it can enforce **safe concurrent executions of SvMs**.
 - If TMOSM knows which ODSSs are used by each SpM as well, then it can enforce **safe concurrent executions of SpMs**.Plus it can enforce **safe concurrency between SvM executions and SpM executions**.



Apr-05 21

UCI
DREAM Lab

Parameter Structure

- An SvM can **accept only a single structured parameter**.
- Since a structured parameter can hold an indefinitely large amount and variety of values, this is not much of a restriction.
- The client node sends the single structured parameter along with the information in the **size** of the parameter (in bytes) to the node hosting the SvM in the form of a message.
 - A particular implementation of the TMO execution engine imposes a limit on the size of the structured parameter that can be used.

Apr-05 22

UCI
DREAM Lab

Ex 1. Definition of an SvM Method Body

- Not calling other SvMs

```
#include "TMOSL.h"
using namespace TMO;
typedef struct {
    MicroSecond system_age;
} ParamStruct_TMO1class_SvM;

class TMO1: public CTMOBase
{
public:
    TMO1 (TCHAR* TMO_external_name, tms TMO_start_time);
private:
    void SvM1 (ParamStruct_TMO1class_SvM * param_struct);
};
```

Apr-05

23

UCI
DREAM Lab



Ex 1. Definition of an SvM Method Body (cont)

- Not calling other SvMs

```
void TMO1::SvM1 (ParamStruct_TMO1class_SvM * param_struct)
{
    MicroSec client_system_age = param_struct->system_age;
    TMOSLprintf (_T("client_system_age");
    TMOSLprintf (_T("time %I64d is issued at %I64d\n"),
        client_system_age);
}
```

Apr-05

24

UCI
DREAM Lab



Call Timestamp

- A call by a client for an SvM involves generation of a **call timestamp**, also called an **SRtimestamp**, on the node hosting the client.
- The call timestamp is carried as a part of the service call message to the node hosting the SvM.
- Inside the function body of the SvM, the call timestamp can be retrieved by use of API
"Microsec **GetSRTimestamp ()**".

Apr-05 25



Ex 2. SvM using an ODSS and a call timestamp - Main.cpp

```
void main ()
{
    StartTMOengine ();
    tms TMO_start_time0 = tm4_DCS_age (2*1000*1000);
    TMO1 T1 (_T("TMO1"), _T("SvM1"), TMO_start_time0);
    MainThrSleep ();
}
```

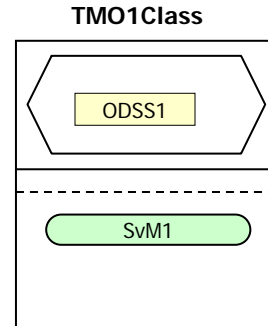
Apr-05 26



Ex 2. SvM using an ODSS and a call timestamp (Cont) - TestTMO1.h

```
typedef struct {
    MicroSecond system_age;
} ParamStruct_TMO1class_SvM1;

class TMO1: public CTMOBase
{
public:
    TMO1 (TCHAR* TMO_external_name,
          tms TMO_start_time);
private:
    ODSSClass1 ODSS1;
    int SvM1 (ParamStruct_TMO1class_SvM1 *);
};
```



Apr-05 27



Ex 2. SvM using an ODSS and a call timestamp (Cont) - TestTMO1.cpp

```
TMO1::TMO1 (TCHAR * TMO_external_name, tms TMO_start_time1)
{
    // register SvM
    SvM_RegistParam svm_spec;
    svm_spec.GETB = 20 * 1000;
    _tcscpy (svm_spec.name, _T("SvM1"));
    svm_spec.build_regist_info_ODSS (ODSS1.GetId (), RO);
    RegisterSvM ( (PFSvMBody) SvM1, & svm_spec);

    // register TMO
    ...
}
```

Apr-05 28



Ex 2. SvM using an ODSS and a call timestamp (Cont) - TestTMO1.cpp

```
int TMO1::SvM1 (ParamStruct_TMO1class_SvM1 * pReq)
{
    // Get the call timestamp created when the SR was issued.
    MicroSec SRTimestamp = GetSRTimestamp ();
    MicroSec client_system_age = pReq->system_age;
    TMOslprintf (_T("SvM1 receives a request with request time of
    %l64d which is issued at %l64d.\n"),
    (__int64) client_system_age,
    SRTimestamp);
}

```

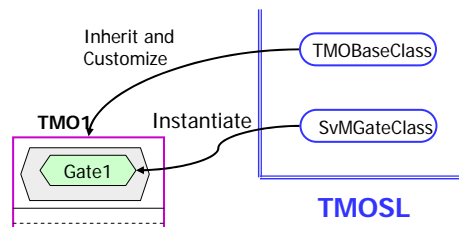
Apr-05 29



Creation of SvM Gates

- A gate object is a front-end interface of an automatically created call-path to a remote SvM.
 - When a gate is instantiated, the symbolic name of the associated SvM is used as a parameter for the constructor.
- Client objects can just use the gate's name in calling the associated remote SvM without using the symbolic name of the SvM.
- The **SvMGateClass** in the TMOsl is a fully defined class.
 - Gates are created by **instantiating** the SvMGateClass in the TMOsl.


```
SvMGateClass Gate1 (_T("TMO3"), _T("SvM7"), tm4_DCS_age(7*1000*1000) )
```
 - Also provides a user-friendly interface for the C++ TMO programmer who needs to make TMOsl perform various kinds of **service calls** to SvMs.



Apr-05 30



Definition of SvMGateClass

```
namespace TMO {
// Environment Access Capability Establishment
class TMO_SL_API SvMGateClass: public ODSSBaseClass <SvMGateClass>
// In TMO_SL
{
protected:
    TCHAR        serverTMO_Name [MAX_TMO_NAME_LEN];
    TCHAR        SvM_Name [MAX_SVM_NAME_LEN];
    tms          service_start_time;
    HANDLE       GateHandle;

public:
    SvMGateClass (const TCHAR* server_TMO_name,
                  const TCHAR* SvM_name1,
                  const tms& service_start_time);

    ~SvMGateClass ();

    const TCHAR* get_TMO_name () const;
    const TCHAR* get_SvM_name () const;
};
```

Apr-05

31

UCI
DREAM Lab



Definition of SvMGateClass (Cont)

```
int BlockingSR ( void* ParamPtr, int ParamSize,
                MicroSec ResponsePeriod, tms ORT );
// Send a service request to the server SvM and wait until the reply returns
// from the server SvM or the ResponsePeriod has expired.
// The "ORT" indicates when this SR message should be released to
// the SvM execution scheduler on the node hosting the called SvM.
// This returns SUCCESS or FAIL.

int BlockingSR ( void* ParamPtr, int ParamSize,
                tms RTDeadline, tms ORT );
// Send a service request to the server SvM and wait until the reply returns
// from the server SvM or the RTDeadline has been reached. The only
// difference between two variants of BlockingSR () is in defining the
// deadline for result arrival as either period (ResponsePeriod) or absolute
// time (RTDeadline).
// This returns SUCCESS or FAIL.
```

Apr-05

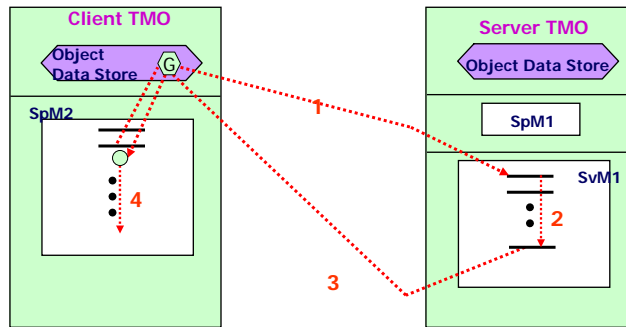
32

UCI
DREAM Lab



Definition of SvMGateClass (Cont)

- int **BlockingSR** (void* ParamPtr, int ParamSize, MicroSec ResponsePeriod, tms ORT)

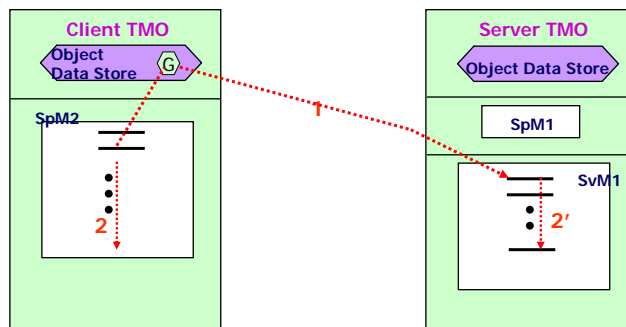


Apr-05 33



Definition of SvMGateClass (Cont)

- Int **OnewaySR** (void * ParamPtr, int ParamSize, tms ORT);
 // Send a service request to the server SVM and the control returns to
 // the caller immediately. **No result can be returned.**
 // This returns SUCCESS or FAIL.



Apr-05 34



Definition of SvMGateClass (Cont)

```

int NonBlockingSR (void * ParamPtr, int ParamSize, tmsp & Timestamp,
                  tms ORT);
    // Send a service request to the server SvM and the control returns to the
    // caller immediately. A timestamp is returned and can be used to check
    // the result of this service request later.
    // This returns SUCCESS or FAIL.

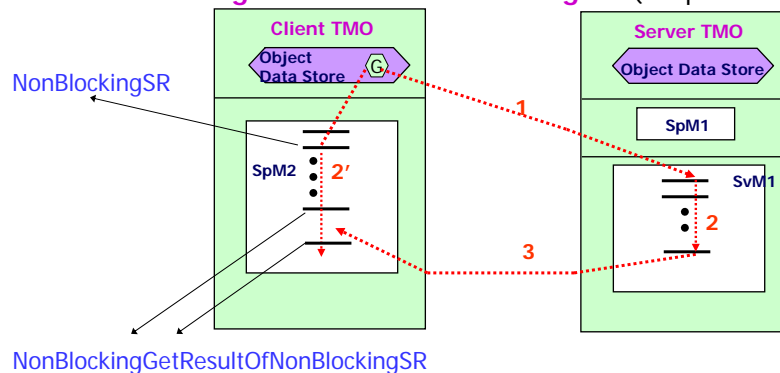
int NonBlockingGetResultOfNonBlockingSR (tmsp Timestamp);
    // Get the reply returned from the server SvM if the reply has arrived.
    // Timestamp is used to search for the reply
    // corresponding to a specific service request made earlier.
    // This returns SUCCESS or FAIL.
    
```

Apr-05 35



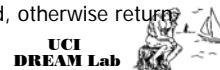
Definition of SvMGateClass (Cont:)

- int NonBlockingSR (void * ParamPtr, int ParamSize, tmsp & Timestamp, tms ORT) along with
- int NonBlockingGetResultOfNonBlockingSR (tmsp Timestamp)



- Get the reply returned from the server SvM if the reply has arrived, otherwise return FAIL.

Apr-05 36



Definition of SvMGateClass(Cont:)

```

int BlockingGetResultOfNonBlockingSR ( tmsp Timestamp,
                                         MicroSec ResponsePeriod);
// Wait until the reply returns from the server SvM or the ResponsePeriod
// has expired.
// Timestamp is used to search for the reply
// corresponding to a specific service request made earlier.
// This returns SUCCESS or FAIL.

int BlockingGetResultOfNonBlockingSR (tmsp Timestamp,
                                         tms RTDeadline);
// Wait until the reply returns from the server SvM or the RTDeadline
// has expired. Timestamp is used to search for the reply
// corresponding to a specific service request made earlier.
// The only difference between two variants of
// BlockingGetResultOfNonBlockingSR () is in defining the
// deadline for result arrival as either period (ResponsePeriod) or absolute
// time (RTDeadline). This returns SUCCESS or FAIL.
    
```

Apr-05 37

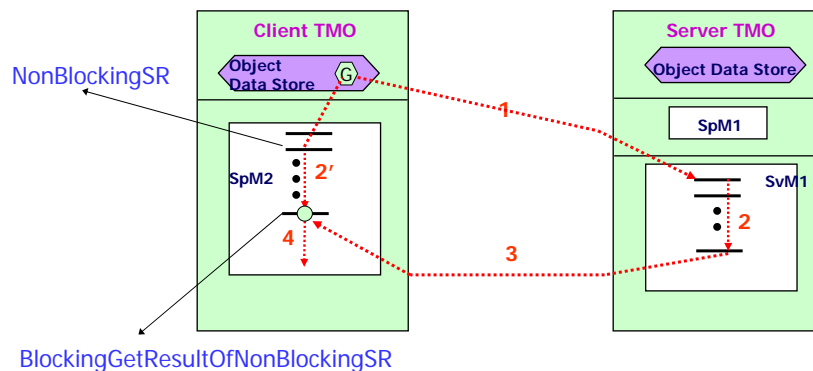


Definition of SvMGateClass (Cont)

- int **NonBlockingSR** (void * ParamPtr, int ParamSize, tmsp & **Timestamp**, tms **ORT**)
- along with
- ```

int BlockingGetResultOfNonBlockingSR (tmsp Timestamp,
 MicroSec ResponsePeriod)

```



- Wait until the reply returns from the server SvM or the ResponsePeriod / RTDeadline has passed.

Apr-05 38



## Definition of SvMGateClass (Cont)

```
int ClientTransferSR (int Client_RRQID,
 void * OrigParamPtr, int OrigParamSize,
 void * AddedParamPtr, int AddedParamSize);
```

Not used in EECS123

```
// Pass a client's request to another SvM. OrigParamPtr is the parameter
// structure that the initial client of the client-transfer call chain supplied.
// The caller of each client-transfer call may pass an additional
// parameter structure, AddedParamPtr, which contains
// intermediate results produced by the caller but not included in
// OrigParamPtr.
// This returns SUCCESS or FAIL.
```

```
};
```

```
}; // End of namespace TMO
```

Apr-05

39

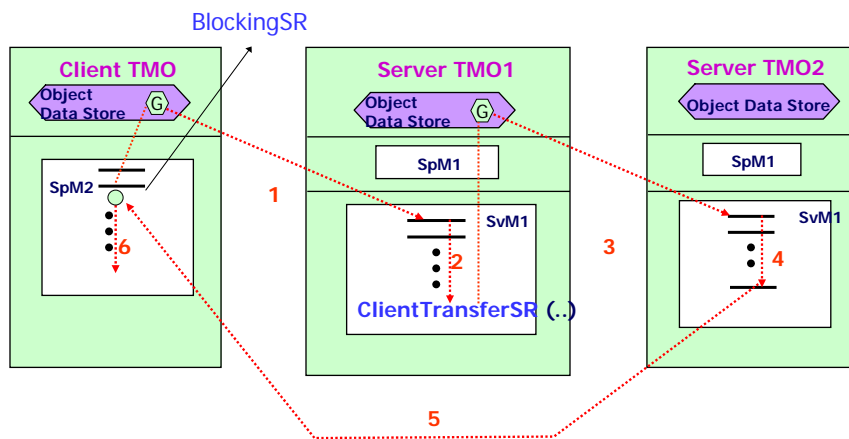
UCI  
DREAM Lab



## Definition of SvMGateClass (Cont)

- ClientTransferSR

Not used in EECS123



Apr-05

40

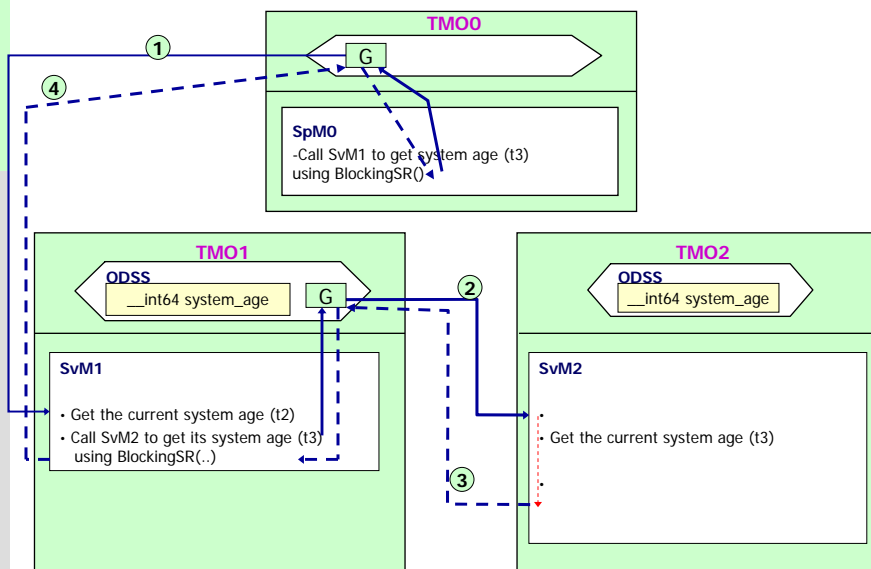
UCI  
DREAM Lab



## An Example Network of TMOs Containing SvMs



### TMO Network Structure



## TypeStructure.h

```
typedef struct {
 __int64 system_age;
} ParamStruct_TMO1class_SvM;
```

```
typedef struct {
 __int64 system_age;
} ParamStruct_TMO2class_SvM;
```

Apr-05

43

UCI  
DREAM Lab



## TestTMO0.h

```
class TMO0: public CTMOBase
{
public:
 TMO0 (TCHAR*, TCHAR*, TCHAR*, tms, tms);
private:
 SvMGateClass gate1;
 int SpM0 ();
};
```

Apr-05

44

UCI  
DREAM Lab



## TestTMO0.cpp

```
int TMO0::SpM0 ()
{
 ParamStruct_TMO1class_SvM para0;
 para0.system_age = GetCurrentDCSage ();
 MicroSec deadline = 100* 1000;
 TMOSSLprintf (_T("SpM0 sends a request at: %I64d.\n"), para0.system_age);
 tms ORT = tm4_DCS_age (GetCurrentDCSage () + 50*1000);
 gate1.BlockingSR (¶0, sizeof(para0), deadline, ORT);
 TMOSSLprintf (_T("SpM0 receives a reply at: %I64d, with a return value:
 %I64d (SvM2 start time).\n=====\\n\\n"),
 GetCurrentDCSage (), para0.system_age);

 return 1;
}
```

Apr-05

45

UCI  
DREAM Lab



## TestTMO0.cpp

```
TMO0::TMO0 (TCHAR * TMO_external_name, TCHAR* ServerTMO_name, TCHAR *
ServerSvM_name, tms TMO_start_time, tms gate_start_time):
 Gate1 (ServerTMO_name, ServerSvM_name, gate_start_time)
{
 // register SpM
 SpM_RegistParam spm_regist_param;
 AAC aac (
 NULL, // null for candidate aac label
 tm4_DCS_age (5*1000*1000),
 tm4_DCS_age (6*1000*1000),
 1000 * 1000,
 0 * 1000,
 200 * 1000,
 250 * 1000
);
}
```

Apr-05

46

UCI  
DREAM Lab



## TestTMO0.cpp

```
spm_regist_param.build_regist_info_AAC (aac);
spm_regist_param.build_regist_info_ODSS (gate1.GetId(), RW);
RegisterSpM ((PFSpMBody) SpM0, & spm_regist_param);

// register TMO
TMO_RegistParam tmo_spec;
_tcscpy (tmo_spec.global_name, TMO_external_name);
tmo_spec.start_time = TMO_start_time;
RegisterTMO (& tmo_spec);
}
```

Apr-05

47

UCI  
DREAM Lab



## TestTMO1.h

```
class ODSSClass1: public ODSSBaseClass <ODSSClass1>
{
public:
 int Count;
 ODSSClass1() { Count = 0; }
};
```

Apr-05

48

UCI  
DREAM Lab



## TestTMO1.h

```
class TMO1: public CTMOBase
{
public:
 TMO1 (TCHAR* , TCHAR* , TCHAR* , TCHAR* , tms , tms);
private:
 ODSSClass1 ODSS1;
 SvMGateClass gate2;
 int SvM1 (ParamStruct_TMO1class_SvM *);
};
```

Apr-05

49

UCI  
DREAM Lab



## TestTMO1.cpp

```
int TMO1::SvM1 (ParamStruct_TMO1class_SvM * param1)
{
 MicroSec SRTimestamp = GetSRTimestamp ();
 TMOSLprintf (_T("SvM1 receives a request with parameter %I64d which is issued at %I64d.\n"), param1->system_age, SRTimestamp);

 ParamStruct_TMO2class_SvM param2;
 param2.system_age = GetCurrentDCSage ();
 MicroSec deadline = 50* 1000;

 TMOSLprintf (_T("SvM1 sends a request at: %I64d.\n"), param2.system_age);
 gate2.BlockingSR (¶m2, sizeof (param2), deadline,
 tm4_DCS_age (GetCurrentDCSage () + 20 * 1000));
 TMOSLprintf (_T("SvM1 receives a reply.\n"));
 param1->system_age = param2.system_age;
 return 1;
}
```

Apr-05

50

UCI  
DREAM Lab



## TestTMO1.cpp

```
TMO1::TMO1 (TCHAR * TMO_external_name,
 TCHAR * SvM_external_name,
 TCHAR * ServerTMO_name,
 TCHAR * ServerSvM_name,
 tms gate_start_time,
 tms TMO_start_time):
 gate2 (ServerTMO_name, ServerSvM_name, gate_start_time)
{
 // register SvM
 SvM_RegistParam svm_spec;
 svm_spec.GETB = 20 * 1000;
```

|        |    |
|--------|----|
| Apr-05 | 51 |
|--------|----|



## TestTMO1.cpp

```
_tcscpy (svm_spec.name, SvM_external_name);
svm_spec.build_regist_info_ODSS (odss1.GetId(), RO);
svm_spec.build_regist_info_ODSS (gate2.GetId(), RW);
RegisterSvM ((PFSvMBody) SvM1, &svm_spec);

// register TMO
TMO_RegistParam tmo_spec;
_tcscpy (tmo_spec.global_name, TMO_external_name);
tmo_spec.start_time = TMO_start_time;
RegisterTMO (& tmo_spec);
}
```

|        |    |
|--------|----|
| Apr-05 | 52 |
|--------|----|



## TestTMO2.h

---

```
class ODSSClass2: public ODSSBaseClass <ODSSClass2>
{
public:
 int Count;
 ODSSClass2 () { Count = 0; }
};
```

Apr-05

53



## TestTMO2.h

---

```
class TMO2: public CTMOBase
{
public:
 TMO2 (TCHAR* , TCHAR* , tms);
private:
 ODSSClass2 ODSS2;
 int SvM2 (ParamStruct_TMO2class_SvM *);
};
```

Apr-05

54



## TestTMO2.cpp

```
int TMO2::SvM2 (ParamStruct_TMO2class_SvM * pReq)
{
 MicroSec SRTimestamp = GetSRTimestamp ();
 MicroSec curtime = GetCurrentDCSage ();
 TMOslprintf (_T("SvM2 is triggered at %l64d with request time of
 %l64d which is issued at %l64d.\n"), curtime, pReq->system_age,
 SRTimestamp);
 pReq->system_age = curtime;
 return 1;
}
```

Apr-05

55

UCI  
DREAM Lab



## TestTMO2.cpp

```
TMO2::TMO2 (
 TCHAR* TMO_external_name,
 TCHAR* SvM_external_name,
 tms TMO_start_time1):
{
 // register SvM
 SvM_RegistParam svm_spec;
 svm_spec.GETB = 20 * 1000;
 _tcscpy (svm_spec.name, SvM_external_name);
 svm_spec.build_regist_info_ODSS (ODSS2.GetId(), RO);
 RegisterSvM ((PFSvMBody) SvM2, &svm_spec);
}
```

Apr-05

56

UCI  
DREAM Lab



## TestTMO2.cpp

```
// register TMO
TMO_RegistParam tmo_spec;
_tcscpy (tmo_spec.global_name, TMO_external_name);
tmo_spec.start_time = TMO_start_time1;
RegisterTMO (& tmo_spec);
}
```

Apr-05

57



## Main.cpp

```
void main ()
{
 StartTMOengine ();
 tms TMO_start_time0 = tm4_DCS_age (2*1000*1000);
 TMO0 T0 (_T("TMO0"), _T("TMO1"), _T("SvM1"),
 TMO_start_time0, tm4_DCS_age (3*1000*1000));
 TMO1 T1 (_T("TMO1"), _T("SvM1"), _T("TMO2"),
 _T("SvM2"), TMO_start_time0,
 tm4_DCS_age(3*1000*1000));
 TMO2 T2 (_T("TMO2"), _T("SvM2"), TMO_start_time0);
 MainThrSleep ();
 return;
}
```

Apr-05

58

