

**EECS123:**

**Introduction to  
Real-Time Distributed Programming**

**Intro to TMO,  
Global Time Representations,  
Spontaneous Method (SpM),  
Object Data Store Segment (ODSS)**

Feb-07	1
--------	---

UCI  
DREAM Lab



**Time-Triggered Actions  
in Single Node Systems  
- TMO Approach**

- Time-Triggered Action
- A Rough Introduction to TMO
- Time Classes in TMO: Global Time Representations
- Spontaneous Method (SpM) in TMO
- Sample Program V (A simple SpM)
- ODSS Class in TMO
- Sample Program VI (A simple ODSS)
- Sample Program VII (A simple TMO)

Feb-07	2
--------	---

UCI  
DREAM Lab



## Time-Triggered Action

- **At time T do S** :
  - A fundamental & distinguishing part of real-time programming
  - If S is a function, a control signal for the activation of the function in a node is derived from the progression of real-time;
  - Whenever the real-time clock within a node reaches a preset value T specified in a scheduling table, a control signal is generated;
  - In principle, S may be a single assignment statement, a compound statement, or a function.

Cf. **Event-triggered** := The control signal is derived from a significant state change, i.e., an event, in the environment or within the computer system .

Feb-07	3
--------	---

UCI  
DREAM Lab



A rough introduction to TMO --

## Facilities in TMO for Time-Triggered Actions

**TMO** := Time-Triggered  
Message-Triggered  
Object

Feb-07	4
--------	---

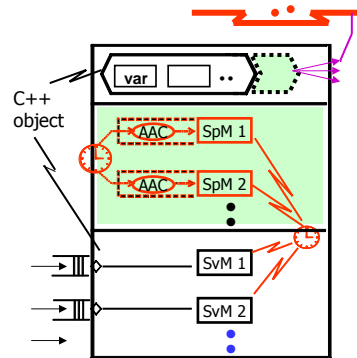
UCI  
DREAM Lab



## New-Generation Real-Time (RT) Distributed Computing (DC) Component

### High-Level RT Object: Time-triggered Message-triggered Object (TMO)

- Initiated in early 1990's
- Meant to be a **natural easy-to-use extension** of the C++/Java object
- Can support **Hard-RT DC Software Engineering (SE)** as well as **Non-RT DC SE**
- Contains **only high-level intuitive** and yet **precise expressions** of timing requirements



UCI  
DREAM Lab

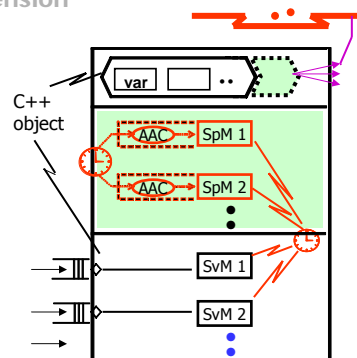


Feb-07 5

## New-Generation RT DC Component

### High-Level RT Object: Time-triggered Message-triggered Object (TMO)

- Meant to be a **natural easy-to-use extension** of the C++/Java object
- Can support **Hard-RT DC SE** as well as **Non-RT DC SE**
- Contains **only high-level intuitive** and yet **precise expressions** of timing requirements
- **Formulated from the beginning with the objective of enabling design-time guaranteeing** of timely actions



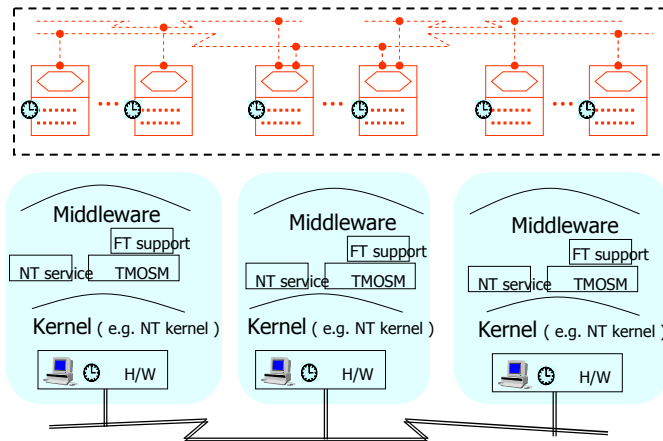
UCI  
DREAM Lab



Feb-07 6

## TMO Network Structuring

All conceivable RT DC applications and Non-RT applications can be Structured as TMO networks .



Feb-07 7

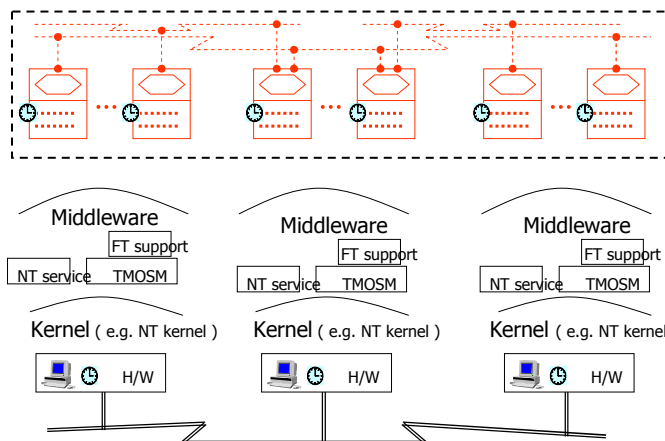
UCI DREAM Lab



## Making Applic Programmers' Life Easier: Structure as TMO networks relying on intelligent execution facilities

L0

### Real-Time Distributed Computing Applications



No concerns with

- Processes & Threads
- Object locations (except in avoiding overloaded nodes)
- Low-level comm protocols

No specification of timing requirements in indirect terms (e.g., priorities)

- Only *start-windows* and *completion deadlines* for object methods and
- *time-windows* for output actions

Feb-07 8

UCI DREAM Lab



## A rough introduction to TMO --

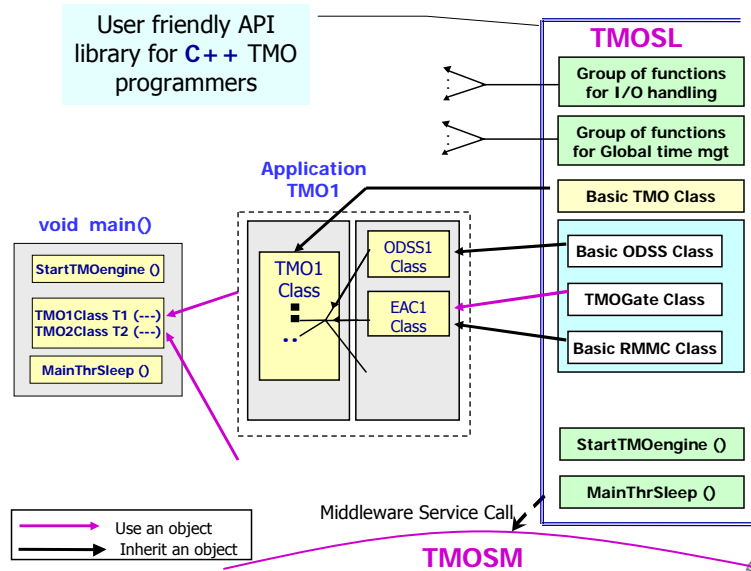
# TMOSL & TMO Toolkit

Feb-07 9

UCI  
DREAM Lab



## TMOSM Support Library (TMOSL)



Feb-07 10

UCI  
DREAM Lab



## How to get TMO toolkit

- To program in TMO, you need the followings:
  - TMOSL user's manual, library & dll files
  - Visual Studio .NET 2003 (preferably) or Visual C++ 6.0
  - Optionally, ConfigAssistant for editing "config.ini" file
- Where to get them
  - Go to the following URL to download **TMO toolkit** including the TMOSL manual, ConfigAssistant, TMOSL header, library & dll files:  
<http://dream.eng.uci.edu/TMOdownload>
  - Send your email address to TA to get a copy of Visual Studio .NET 2003.
- How to program and run your TMO program:
  - For editing config.ini,
    - Refer to the "Appendix A. Computing Station Configuration" of the TMOSL manual
    - ConfigAssistant
  - For setting up your Visual Studio,
    - Refer to the "Appendix G. Getting Started" of the TMOSL manual

Feb-07	11
--------	----

UCI  
DREAM Lab



## A rough introduction to TMO -- TMO Instantiation

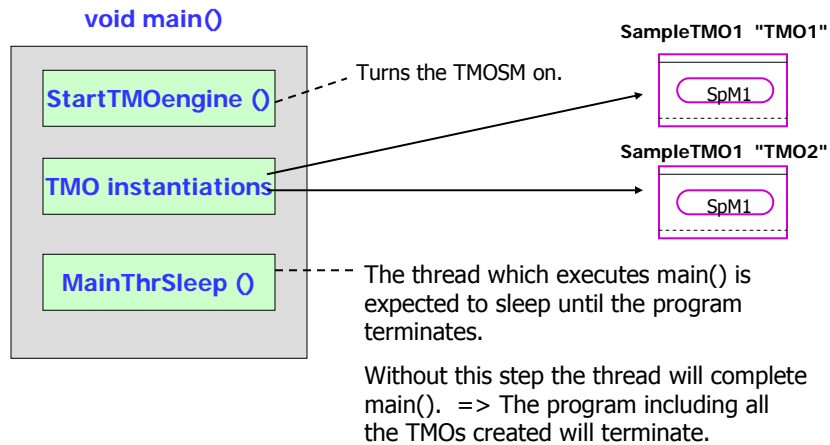
Feb-07	12
--------	----

UCI  
DREAM Lab



## Main () creating a TMO Network

- Main () consists largely of 3 parts:



Feb-07 13

UCI  
DREAM Lab

## Sample Program: TMO instantiation

```
#include "TMOSL.h" ← Include TMOSL header file.
#pragma comment (lib, "TMOSL") ← Link TMOSL library file.
using namespace TMO; ← Use namespace "TMO".

void main()
{
    // Turn on the TMOSM.
    StartTMOengine ();

    // Instantiate TMOs.
    SampleTMO1 T1 ( _T("TMO1"), tm4_DCS_age (2 * 1000 * 1000));
    SampleTMO1 T2 ( _T("TMO2"), tm4_DCS_age (2 * 1000 * 1000));

    // Have the main thread sleep.
    MainThrSleep ();
}

```

a macro for Unicode support

Feb-07 14

UCI  
DREAM Lab

## Sample Program: TMO instantiation (cont.)

```
// SampleTMO class definition
class SampleTMO1 : public CTMOBase
{
public:
    SampleTMO1 (TCHAR *, tms);
    ~SampleTMO1 ();
private:
    // SpM
    void          SpM1 ();
    // ODSS
    ODSSClass     ODSS1;
};
```

Details about SpM and ODSS follow later in this presentation.

Feb-07 15

UCI  
DREAM Lab



## A rough introduction to TMO -- Level-1 Overview of TMO

Feb-07 16

UCI  
DREAM Lab



## What is TMO ?

- Time-Triggered Message-Triggered Object
- C++ TMO
  - A type of a C++ object
  - Can be built in Standard C++ plus an API (Application Programming Interface) library called **TMO SL (TMO Support Library)**
- TMO is a **high-level real-time computing object**
  - Member functions (i.e., methods) are executed within **specified time windows**
  - Timing requirements are specified in **natural intuitive forms** with no esoteric styles imposed

Feb-07 17

UCI  
DREAM Lab



## What is TMO ?

- TMO is a **high-level distributed computing object**
  - TMOs interact among themselves via **remote method calls**.
  - Remote method calls are made in forms that are essentially the same as those of calling conventional object methods.
  - Low-level communication protocols such as TCP/IP are transparent to TMO programmers.
  - Each TMO instantiation has a globally unique name.
  - Moreover, all time references in TMO are **global time references**.

Feb-07 18

UCI  
DREAM Lab



## What is TMO ? (cont.)

- TMO is an **autonomous active object**.
  - TMO can incorporate a new group of member functions called **Spontaneous Methods (SpMs)** (also called **Time-Triggered Methods**).
  - An SpM has an **AAC (Autonomous Activation Condition)**, which is a specification of the time-windows for execution of the SpM.
 

E.g., {  
 "start-during (10am, 10:05am) finish-by 10:10am",  
 "for t = from 10am to 10:50am every 30min  
 start-during (t, t+5 min) finish-by t+10min"  
 }.
  - An SpM is in effect a precisely and flexibly timed **thread**.  
 An SpM generates **livelihood**.  
 An SpM is like a heart in a human body.

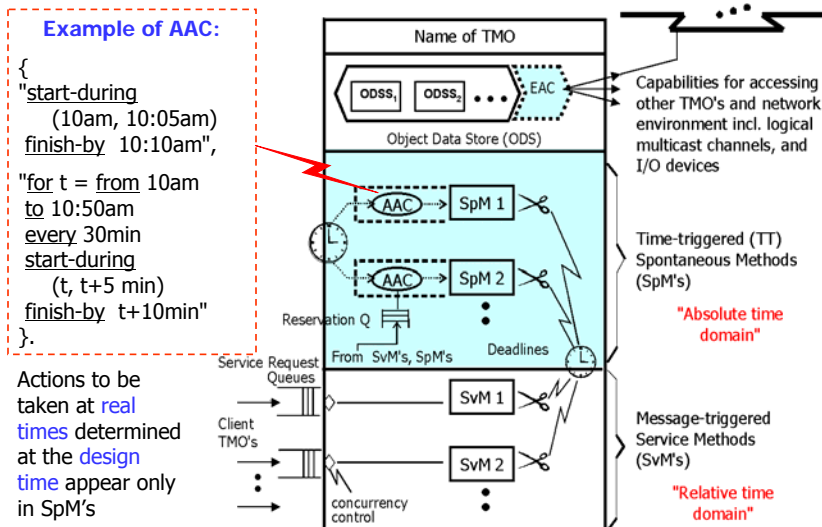
Feb-07 19

UCI  
DREAM Lab



## Spontaneous Methods (SpM's)

L0



Feb-07 20

UCI  
DREAM Lab



## What is TMO ? (cont.)

---

- Any distributed computing application can be built in the form of a TMO network instantiated by main (), nothing else.
  - Partly because an SpM has the power of a thread and more.
- TMOs can also interact via logical multicast channels called **RMMCs** (Real-Time Multicast and Memory Replication Channels).

Feb-07	21
--------	----

UCI  
DREAM Lab



## Global Time Representations in TMO

Feb-07	22
--------	----

UCI  
DREAM Lab



## Time Classes in TMO (Time-Triggered Message-Triggered Object)

- TMOSM (TMO Support Middleware) and TMOSL use a microsecond as the basic time-unit but programmers are advised to check manuals associated with particular implementations of TMOSM to figure out the smallest unit of time that can be used .
- `MicroSec` type is an integer representing the # of microseconds. `C21_age` type is the same as `MicroSec` type, except that it represents the # of microseconds past Jan 1, 2000 in UTC.
- `tms` class is the wrapper class of `C21_age` type, and `tml` class is derived from `tms` class and represents `calendar time`.
- TMOSM records `DCS (Distributed Computing System) age` which represents the # of microseconds lapsed since TMOSM started.
- `DCS age` is returned as timestamp of a non-blocking call when the call is made. Such timestamp is represented by `tmstp` class which includes `DCS age` and `local Node ID`.

Feb-07

23

UCI  
DREAM Lab

## Time Classes in TMO

```
namespace TMO {
typedef __int64 MicroSec;
typedef __int64 C21_age;
    // The number of microseconds past 2000.01.01.00.00.00.000.000
class tms {    // Time-Short
protected:
    C21_age  tm_rep;
public:
    tms (const C21_age &  x = 0);
    tms (const tms &  x);    // Constructors.
    void set_C21_age (const C21_age &  x);
    C21_age get_C21_age () const;
    void increment (const MicroSec &  micsec1);
        // Increment tms instance by given value.
    void tms_to_SYSTEMTIME ( SYSTEMTIME &  st) const;
    void SYSTEMTIME_to_tms ( const SYSTEMTIME &  st);
        // Convert a tms structure to a Windows SYSTEMTIME structure.
};
```

Feb-07

24

UCI  
DREAM Lab

## Time Classes in TMO

```

class tml : public tms // Time-Long {
private:
    int year;           int month;           int day;
    int hour;          int minute;        int second;
    int millisecond;   int microsecond;
    void tms_to_tml ();
    // Similar to tms_to_SYSTEMTIME() but tailored for tml
    void tml_to_tms ();
    // Similar to SYSTEMTIME_to_tms() but tailored for tml
public:
    tml (int year1 = 2000, int month1 = 1, int day1 = 1,
         int hour1 = 0, int minute1 = 0, int second1 = 0,
         int millisecond1 = 0, int microsecond1 = 0);
    tml (const tms& tms1);
    tml (const C21_age& c);
    // Constructors of tml class.
    tms to_tms ();
    // Return tms value.
    void print ();
    // Print out the time values in the form of C21_age and
    // tml (calendar time) type.
}

```

Feb-07

25

UCI  
DREAM Lab

## Time Classes in TMO - Global Functions

```

MicroSec duration (const tms & tm1, const tms & tm2);
    // Return tm2 - tm1.
tms later (const tms & tms1, const MicroSec & micsec1);
    // Return tms1 + micsec1.
tms tm4_C21_age (const C21_age & x);
    // Return tms value converted from C21_age variable.
MicroSec GetCurrentDCSage ();
    // Return current DCS age.
tms TMOSL_API now ();
    // Return the current real time.
tms tm4_DCS_age (const MicroSec & x);
    // Return DCS_start_time + given value.
MicroSec to_DCS_age (const tms & y);
    // Return given value - DCS_start_time.

```

Feb-07

26

UCI  
DREAM Lab

## Time Classes in TMO - Global Functions (cont.)

```

tml to_tm1 (const tms & tms1);
tml to_cal (const tms & tms1);
    // Both of to_tm1 () and to_cal () are identical / synonymous
    // Return tml value converted from given value.
tms tm4_cal (int year, int month, int day, int hour, int minute, int second, int
millisecond, int microsecond);
    // Return tms value converted from given calendar time.
void wait_for (const MicroSec & period);
    // Suspend the execution of the caller's thread for given time.
void wait_until (const tms & r_time);
    // Suspend the execution of the caller's thread until given time.

```

Feb-07

27

UCI  
DREAM Lab

## Time Classes in TMO

```

class tmstp : public tms {           // Timestamp (tmstp) class
private:
    int          NodeID;
public:
    tmstp ();
    tmstp (int node_id, const C21_age & t = 0);
    tmstp (const tmstp & tp);
    // Get and set functions of member variables.
    tmstp get () const;
    void set (int node_id, const C21_age & t = 0);
    void set (const tmstp & tp);
    int  get_node_id () const;
    void set_node_id (int node_id);
};
    --- All remaining parts of TMOSL ---
}; // end of namespace TMO

```

Feb-07

28

UCI  
DREAM Lab

## An example using time classes in TMO

```
// Sometimes need to decompose assignment of
// microsecond-based time value
// because MS compiler limits constant to 32 bits.
MicroSec from = 5;
    from *= 1000 * 1000; // 5 sec
MicroSec until = 2 * 60 * 60;
    until *= 1000 * 1000; // 2 hours
MicroSec every = 1 * 1000 * 1000; // 1 sec
MicroSec est = 0;          MicroSec lst = 15 * 1000; // 15 msec
MicroSec by = 100 * 1000; // 100 msec

AAC AAC1 ( "", // No need to know the exact nature
    tm4_DCS_age(from), // of this for now
    tm4_DCS_age(until),
    every, est, lst, by);

build_regist_info_AAC (AAC1);
```

Feb-07

29

UCI  
DREAM Lab

## Spontaneous Methods (SpMs) in TMO

Feb-07

30

UCI  
DREAM Lab

## Spontaneous Methods (SpMs)

- Time-Triggered Methods
- An SpM has an AAC (Autonomous Activation Condition), which is a specification of the time-windows for execution of the SpM.
  - E.g.,
 

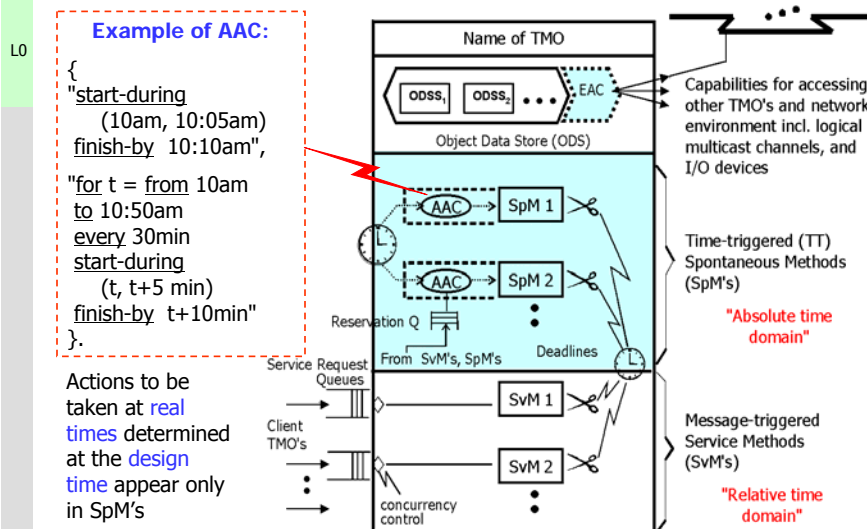
```
{
"start-during (10am, 10:05am) finish-by 10:10am",
"for t = from 10am to 10:50am every 30min
start-during (t, t+5 min) finish-by t+10min"
}.
```
- An SpM is in effect a precisely and flexibly timed thread.
  - An SpM generates livelihood.
  - An SpM is like a heart in a human body.

Feb-07 31

UCI  
DREAM Lab



## Spontaneous Methods (SpM's)

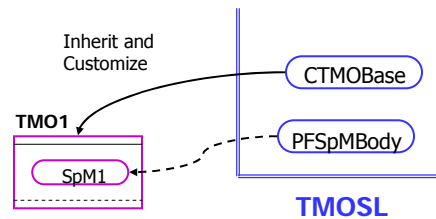


Feb-07 32

UCI  
DREAM Lab



## Creation of a SpM



- SpMs are declared as ordinary member functions of a TMO class.
- Each SpM should be registered to TMOSM with :
  - The set of the names and access modes of the ODSSs which it intends to use
  - AACs, which are specifications of the time-windows for execution of the SpM.

Feb-07	33
--------	----

UCI  
DREAM Lab



## Creation of a SpM (cont.)

```
enum access_mode_type { RO, RW };
struct ODSS_n_access_mode // An ODSS is a group of data
{                          // members.
  int ODSS_ID;
  access_mode_type access_mode; // Read-write or Read-only
};
```

```
class AAC
{
private:
  TCHAR cand_label [MAX_AAC_NAME_SIZE];
  // SpM execution starts at 'from' time and
  // finishes at 'until' time.
  tms from;
  tms until;
```

Feb-07	34
--------	----

UCI  
DREAM Lab



## Creation of a SpM (cont.)

```
// Invoked by TMOSM every 'every' microsec
MicroSec every;
// Earliest allowed Start-Time of SpM in each exec. Round
MicroSec est;
// Latest allowed Start-Time of SpM in each exec. Round
MicroSec lst; // Latest Start Time
```

\* [Est, Lst] is the start-time window. This window must be wider than a minimum size determined for each TMO execution engine. The information on the minimum size should be in Config.ini.

```
// Guaranteed completion time (GCT) or execution duration bound for the SpM
// in each exec. round.
// It is important to declare a tight value, preferably a value much smaller
// than the inter-invocation interval, for this GCT.
// If an unnecessarily large value is given, then the opportunities
// for large SvMs to execute diminish. This is because of
// the rule, BCC (Basic Concurrency Constraint).
MicroSec by;
```

Feb-07

35

UCI  
DREAM Lab

## Creation of an SpM (cont.)

```
public:
AAC (const AAC &);

AAC (const TCHAR* cand_label = _T(""),
    const tms & from1,    const tms & until1,
    const MicroSec & every1, const MicroSec & EST1,
    const MicroSec & LST1, const MicroSec & by1
);
}; // End of AAC
```

Feb-07

36

UCI  
DREAM Lab

## Creation of a SpM (cont.)

```

struct SpM_RegistParam
{
    std::list<ODSS_n_access_mode*> ODSS_access_mode_list;
    std::list<AAC*> AAC_list;
    void build_regist_info_ODSS (int odss_id, access_mode_type mode);
    void build_regist_info_AAC (const AAC & aac_spec);
    ~SpM_RegistParam ();
};

class CTMOBase
{ ...
    // Register an SpM to the middleware
    bool RegisterSpM (PFSpMBody, SpM_RegistParam *);
    ...
};

```

Feb-07

37

UCI  
DREAM Lab

## Sample Program IV (A Simple SpM)

```

#include "TMOSL.h"

class TMO1: public CTMOBase
{
public:
    TMO1 (TCHAR *, AAC &, tms);
private:
    ODSSClass1          ODSS1;
    void                SpM1 ();
};

void TMO1::SpM1 ()
{
    TMOSLprintf ("hello, world!\n");
}

```

Feb-07

38

UCI  
DREAM Lab

## Sample Program IV (A Simple SpM)

```

TMO1::TMO1 (TCHAR* TMO_external_name,
            AAC & aac_spec,
            tms TMO_start_time)
{
    // Preparation of the registration parameters related to SpM
    SpM_RegistParam spm_spec;
    spm_spec.build_regist_info_AAC (aac_spec);
    spm_spec.build_regist_info_ODSS (ODSS1.GetId(), RW);
    RegisterSpM ( (PFSpMBody) SpM1, & spm_spec);

    // Register this TMO to TMOSM
    TMO_RegistParam tmo_spec;
    _tcscpy(tmo_spec.global_name, TMO_external_name);
    tmo_spec.start_time = TMO_start_time1;
    RegisterTMO(&tmo_spec);
}

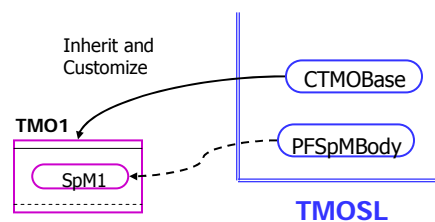
```

Feb-07

39

UCI  
DREAM Lab

## Creation of an Enclosing TMO



- The **CTMOBase** in TMOSL provides a skeleton of a TMO.
  - The TMO programmer can easily create an application-specific TMOClass by inheriting the CTMOBase in the TMOSL.
  - "**RegisterTMO**" must be executed after all the initiation of a TMO is done including initialization of ODSS, SpM, and SvM.

Feb-07

40

UCI  
DREAM Lab

## CTMOBase

```

class CTMOBase
{
public:
    virtual ~CTMOBase ();
    // Register an TMO to the execution engine
    bool RegisterTMO (TMO_RegistParam *);

protected:
    CTMOBase (void);
    // Register an SpM/SvM to the middleware
    bool RegisterSpM (PFSpMBody, SpM_RegistParam *);
    bool RegisterSvM (PFSvMBody, SvM_RegistParam *);
};

```

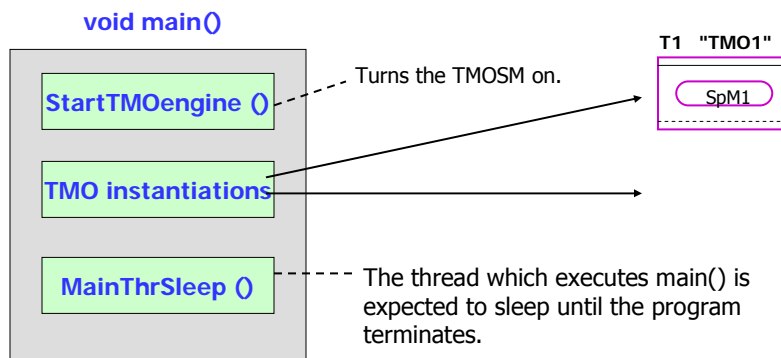
Feb-07

41

UCI  
DREAM Lab

## Main () creating a TMO Network

- Main () consists largely of 3 parts:



Without this step the thread will complete main(). => The program including all the TMOs created will terminate.

Feb-07

42

UCI  
DREAM Lab

## Sample Program IV (A Simple SpM)

- To compile and run TMO programs, users have to include one TMOSL header file and link TMOSL library & dll files:
  - Header file: [TMOSL.h](#)
  - Library files: [TMOSM.lib](#) & [TMOSM.dll](#)
- Sample program

```
// SampleTMO.cpp
#include "TMOSL.h" ← Include TMOSL header file here.
...
#pragma comment (lib, "TMOSM") ← Link TMOSL library file here.
using namespace TMO; ← Use namespace "TMO".
int main ()
{
    StartTMOengine ();
    ...
    MainThrSleep ();
    return 0;
}
```

Feb-07

43

UCI  
DREAM Lab

## Sample Program IV (A Simple SpM)

```
TMO1::TMO1 (TCHAR * TMO_external_name,
            AAC & aac_spec,
            tms TMO_start_time)
{
    // Preparation of the registration parameters related to SpM
    SpM_RegistParam spm_spec;
    spm_spec.build_regist_info_AAC (aac_spec);
    spm_spec.build_regist_info_ODSS (ODSS1.GetId(), RW);
    RegisterSpM ( (PFSpMBody)SpM1, &spm_spec);

    // Register this TMO to TMOSM
    TMO_RegistParam tmo_spec;
    _tcscopy (tmo_spec.global_name, TMO_external_name);
    tmo_spec.start_time = TMO_start_time;
    RegisterTMO (& tmo_spec);
}
```

Feb-07

44

UCI  
DREAM Lab

## Sample Program IV (A Simple SpM)

```

void main ()
{
    StartTMOengine ();
    tms TMO_start_time0 = tm4_DCS_age (2 * 1000 * 1000);
    AAC aac1 (    _T(""),                                //cand_label[]
                tm4_DCS_age (3 * 1000 * 1000), //from
                tm4_DCS_age (60 * 1000 * 1000), //until
                1 * 1000 * 1000,                //every
                0,                               //EST
                50 * 1000,                       //LST
                200 * 1000 );                    //by
    TMO1 T1 (_T("TMO1"), aac1, TMO_start_time0);
    MainThrSleep ();
}

```

Feb-07

45

UCI  
DREAM Lab

## Sample Program V: SpM

```

// SampleTMO1 class definition
class SampleTMO1 : public CTMOBase
{
public:
    SampleTMO1 (TCHAR *, tms);
    ~SampleTMO1 ();
private:
    // SpM
    void          SpM1 ();

    // ODSS
    ODSSClass     ODSS1;
};

```

Feb-07

46

UCI  
DREAM Lab

## Sample Program V: SpM (cont.)

```
// function body of SpM1
void SampleTMO1::SpM1 ()
{
    TMOSLprintf (_T("[%d] This is SpM1!\n"), ODSS1.Count);
    ODSS1.Count ++;
}
```

TMOSLprintf is a special print function provided by TMOSM. You are highly recommended to use this API rather than the generic C/C++ printf.

Feb-07

47

UCI  
DREAM Lab

## Sample Program V: SpM (cont.)

```
// constructor of SampleTMO class
SampleTMO1::SampleTMO1 (TCHAR * TMO_name,
                        tms    TMO_start_time)
{
    // AAC (Autonomous Activation Condition)
    MicroSec    from = 2 * 1000 * 1000;    // 2 sec
    MicroSec    until = 60 * 60;
    until *= 1000 * 1000;                // 1 hour
    AAC aac1 (    NULL,                    // cand_label[]
               tm4_DCS_age (from),        // from
               tm4_DCS_age (until),      // until
               1 * 1000 * 1000,          // every = 1 sec
               0,                        // EST = 0
               50 * 1000,                 // LST = 50 msec
               200 * 1000 );              // by = 200 msec
}
```

} relative values  
to the start of  
each period

Feb-07

48

UCI  
DREAM Lab

## Sample Program V: SpM (cont.)

```

// Registration parameters related to SpM
SpM_RegistParam spm_spec;
spm_spec.build_regist_info_AAC (aac_spec);
spm_spec.build_regist_info_ODSS (ODSS1.GetId (), RW);
RegisterSpM ( (PFSpMBody) SpM1, & spm_spec);

// Register this TMO to TMOSM
TMO_RegistParam tmo_spec;
_tcscpy (tmo_spec.global_name, TMO_name);
tmo_spec.start_time = TMO_start_time;
RegisterTMO (& tmo_spec);
}

```

Details about ODSS follow on the next sample program.

function body of "SpM1"

Feb-07

49

UCI  
DREAM Lab

## Sample Program V: SpM (cont.)

```

#include "TMOSL.h"
#pragma comment (lib, "TMOSL")
using namespace TMO;

void main()
{
    // Turn on the TMOSM.
    StartTMOengine ();

    // Instantiate TMOs.
    SampleTMO1 ( _T("TMO1"), tm4_DCS_age (2 * 1000 * 1000));
    SampleTMO1 ( _T("TMO2"), tm4_DCS_age (2 * 1000 * 1000));

    // Have the main thread sleep.
    MainThrSleep ();
}

```

Include TMOSL header file.

Link TMOSL library file.

Use namespace "TMO".

a macro for Unicode support

Feb-07

50

UCI  
DREAM Lab

## SpM with ODSSs

Feb-07

51

UCI  
DREAM Lab

## Object Data Store Segment (ODSS)

- A group of data members.
- The basic unit of storage which can be locked for **exclusive access** by a certain TMO method execution or for **shared use in read-only mode** by multiple concurrent executions of TMO methods (SpMs or SvMs).
- Contains a part of the state information of the enclosing TMO.
- When a TMO method (SpM or SvM) is invoked, all the ODSSs to be accessed during that method execution are **locked** before the execution begins.
- During the construction of a TMO method (SpM or SvM), the names of all the ODSSs to be accessed during that method execution are **registered** with TMOSM.

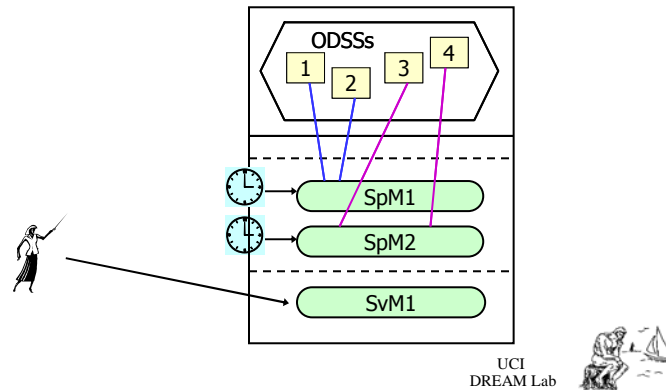
Feb-07

52

UCI  
DREAM Lab

## Why register ODSSs ?

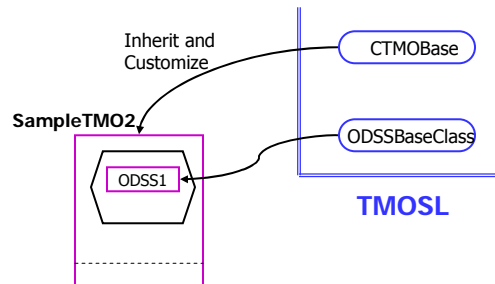
- If TMOSM knows which ODSSs are used by each SpM, then it can enforce **safe concurrent executions of SpMs**.
  - If TMOSM knows which ODSSs are used by each SvM as well, then it can enforce **safe concurrent executions of SvMs**.
- Plus it can enforce **safe concurrency between SpM executions and SvM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



## ODSS (cont)

- The designer of a TMO method may insert a call for "ReleaseODSS ()" in the function body of the TMO method where it is obvious that the associated ODSS will not be accessed during the remainder of the method execution.
  - Such an early release will enable earlier initiation of some other TMO method executions than in the cases where such early releases are not exercised.
  - Once a TMO method execution releases an ODSS early via ReleaseODSS (), it cannot lock the ODSS again before it terminates.

## Creation of an ODSS & an Enclosing TMO



Feb-07	55
--------	----

UCI  
DREAM Lab



## ODSSBaseClass

- The ODSSBaseClass in TMOSL serves as a user-friendly interface for C++ TMO programs that call for TMOSM services related to ODSS programming.
- The TMO programmer should first define an application-specific *ODSSClass* by inheriting the ODSSBaseClass in TMOSL.
- However, the ODSSBaseClass is actually defined as a template class in TMOSL which has a parameter and must be used in the following style:

```
class ODSS1Class : public ODSSBaseClass <ODSS1Class>
{
    public: ---
}
```

Feb-07	56
--------	----

UCI  
DREAM Lab



## ODSSBaseClass

---

```

class ODSSBaseClass {
private:
    int ODSSID; // ID of ODSS
public:
    // Call TMOSM service to register ODSS and save
    // the returned ODSSID in ODSSID.
    ODSSBaseClass ();

    // Call TMOSM service to deregister ODSS.
    ~ODSSBaseClass ();

    // Return this ODSS's ID.
    int GetId ();
  
```

Feb-07

57

UCI  
DREAM Lab

## ODSSBaseClass

---

```

    // The calling TMO method releases the lock
    // on the ODSS since it no longer needs to access
    // the ODSS. So, it cannot lock the ODSS again
    // before it terminates.
    // By releasing its ODSSs early, it may allow
    // other TMO method executions to start early.
    // To enable pipelined executions of the same
    // SvM, early release of ODSSs in the SvM is
    // essential.
    // Return value is SUCCESS or FAIL.
    int ReleaseODSS ();
  
```

Feb-07

58

UCI  
DREAM Lab

## ODSSBaseClass

```
// The calling TMO method changes the Read-Write
// (RW) lock that it has held
// on the ODSS into the Real-Only (RO) lock.
// This change cannot be reversed
// before the calling TMO method terminates.
// By changing the lock this way, it may allow
// other TMO method executions to start early.
// To enable pipelined executions of the same
// SvM, early release or at least this lock
// change of ODSSs in the SvM is
// essential.
// Return value is SUCCESS or FAIL.
int RW2RO ();
};
```

Feb-07

59

UCI  
DREAM Lab

## Sample Program VI-a (A Simple ODSS)

```
// Example of using ODSSBaseClass
// User should define an ODSSClass by inheriting the ODSSBaseClass as
// follows
class ODSSClass1: public ODSSBaseClass <ODSSClass1>
{
    public:
        // ODSS Data members to be defined by the user
        int temperature;
        ---
        // Constructor
        ODSSClass1 (void) { temperature = 0; }
}
```

Feb-07

60

UCI  
DREAM Lab

## Sample Program VI-b (A Simple ODSS)

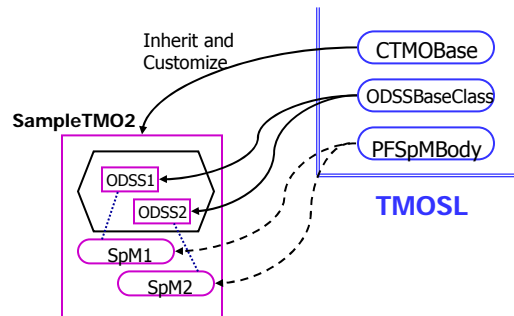
```
// An extra-cautious programmer may define it as follows.
// However, this style is not recommended for most cases.
class ODSSClass1: public ODSSBaseClass <ODSSClass1>
{
    private:
        // ODSS Data structures to be defined by the user
        int temperature;
        ---
    public:
        // Constructor
        ODSSClass1 (void)          { temperature = 0; }
        // ODSS access functions to be defined by the user
        void GetTemperature (int* i) { *i = temperature; }
        void SetTemperature (int i)  { temperature = i; }
        ---
};
```

Feb-07 61

UCI  
DREAM Lab



## Creation of a TMO containing 2 SpMs



Feb-07 62

UCI  
DREAM Lab



## Sample Program VII (A Simple TMO application)

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

Feb-07

63

UCI  
DREAM Lab



## Sample Program VII (A Simple TMO application)

```
class TMO1: public CTMOBase
{
public:
    TMO1 (TCHAR *, AAC &, tms);
private:
    ODSSClass ODSS1;
    ODSSClass ODSS2;
    void SpM1 ();
    void SpM2 ();
};
```

Feb-07

64

UCI  
DREAM Lab



## Sample Program VII (A Simple TMO application)

```
void TMO1::SpM1 ()
{
    TMOSLprintf (" [%d] This is SpM1!Wn", ODSS1.Count);
    ODSS1.Count ++;
}

void TMO1::SpM2 ()
{
    TMOSLprintf (" [%d] This is SpM2!Wn", ODSS2.Count);
    ODSS2.Count ++;
}
```

Feb-07

65

UCI  
DREAM Lab

## Sample Program VII (A Simple TMO application)

```
TMO1::TMO1 (TCHAR * TMO_external_name, AAC & aac_spec, tms TMO_start_time)
{
    // Preparation of the registration parameters related to SpM
    SpM_RegistParam spm_spec1;
    spm_spec1.build_regist_info_AAC (aac_spec);
    spm_spec1.build_regist_info_ODSS (ODSS1.GetId(), RW);
    RegisterSpM ( (PFSpMBody) SpM1, & spm_spec1);

    SpM_RegistParam spm_spec2;
    spm_spec2.build_regist_info_AAC (aac_spec);
    spm_spec2.build_regist_info_ODSS (ODSS2.GetId(), RW);
    RegisterSpM ( (PFSpMBody) SpM2, & spm_spec2);

    // Register this TMO to TMOSM
    TMO_RegistParam tmo_spec;
    _tcscopy (tmo_spec.global_name, TMO_external_name);
    tmo_spec.start_time = TMO_start_time1;
    RegisterTMO (& tmo_spec);
}
```

Feb-07

66

UCI  
DREAM Lab

## Sample Program VII (A Simple TMO application)

```

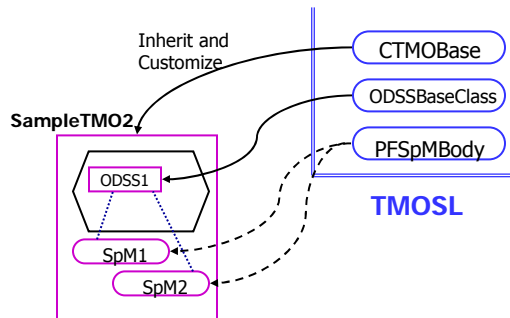
void main ()
{
    StartTMOengine ();
    tms TMO_start_time0 = tm4_DCS_age (2 * 1000 * 1000);
    AAC aac1 (    _T(""),                                //cand_label[]
               tm4_DCS_age (3 * 1000 * 1000),         //from
               tm4_DCS_age (60 * 1000 * 1000),         //until
               1 * 1000 * 1000,                         //every
               0,                                       //EST
               50 * 1000,                               //LST
               200 * 1000 );                           //by
    TMO1 T1 (_T("TMO1"), aac1, TMO_start_time0);
    MainThrSleep ();
}
    
```

Feb-07 67

UCI  
DREAM Lab



## Creation of a TMO with ODSS Sharing



Feb-07 68

UCI  
DREAM Lab



## Sample Program VIII: TMO with ODSS Sharing

```
// SampleTMO2.h
#pragma once ← Ask compiler to include this header file only once.
#include "TMO2L.h"
#pragma comment(lib, "TMO2L")
using namespace TMO;

// User-defined ODSS class definition
class ODSSClass: public ODSSBaseClass <ODSSClass>
{
public:
    // Note that the data member is declared as public.
    int Count;

    // constructor
    ODSSClass ()
    {
        Count = 0;
    }
};
```

Feb-07

69

UCI  
DREAM Lab

## Sample Program VIII: TMO with ODSS Sharing (cont.)

```
// Application TMO class with 2 SpMs and one ODSS shared by the 2 SpMs
class SampleTMO2 : public CTMOBase
{
public:
    // constructor
    SampleTMO2 (TCHAR *, tms);
private:
    // user-defined ODSS
    ODSSClass ODSS1; // shared by SpM1 & SpM2

    // SpM
    void SpM1 ();
    void SpM2 ();
};

// end of SampleTMO2.h
```

Feb-07

70

UCI  
DREAM Lab

## Sample Program VIII: TMO with ODSS Sharing (cont.)

```

SampleTMO2::SampleTMO2 (TCHAR* TMO_name, tms TMO_start_time)
{
    // AACs for both SpM1 & SpM2
    MicroSec      from = 2 * 1000 * 1000;
    MicroSec      until = 60 * 1000 * 1000;
    AAC aac1 (    NULL,                                // cand_label[]
               tm4_DCS_age (from),                    // from
               tm4_DCS_age (until),                  // until
               1 * 1000 * 1000,                       // every
               10 * 1000,                             // EST
               50 * 1000,                             // LST
               500 * 1000 );                          // by
    AAC aac2 (    NULL,                                // cand_label[]
               tm4_DCS_age (from),                    // from
               tm4_DCS_age (until),                  // until
               1 * 1000 * 1000,                       // every
               100 * 1000,                            // EST
               150 * 1000,                            // LST
               200 * 1000 );                          // by
}

```

Feb-07

71

UCI  
DREAM Lab

## Sample Program VIII: TMO with ODSS Sharing (cont.)

```

// Register SpM1 to TMOSM
SpM_RegistParam spm_spec1;
spm_spec1.build_regist_info_AAC (aac1);
spm_spec1.build_regist_info_ODSS (ODSS1.GetId (), RW);
RegisterSpM ( (PFSpMBody) SpM1, & spm_spec1);

// Register SpM2 to TMOSM
SpM_RegistParam spm_spec2;
spm_spec2.build_regist_info_AAC (aac2);
spm_spec2.build_regist_info_ODSS (ODSS1.GetId (), RW);
RegisterSpM ( (PFSpMBody) SpM2, & spm_spec2);

// Register this TMO to TMOSM
TMO_RegistParam tmo_spec;
_tcscpy (tmo_spec.global_name, TMO_name);
tmo_spec.start_time = TMO_start_time;
RegisterTMO (& tmo_spec);
}

```

Register this SpM (SpM1)  
as a writer of ODSS1.

Register this SpM (SpM2)  
as a writer of ODSS1.

Feb-07

72

UCI  
DREAM Lab

## Sample Program VIII: TMO with ODSS Sharing (cont.)

```

// SampleTMO2.cpp
#include "SampleTMO2.h"

// main function of this application
void main()
{
    StartTMOEngine ();

    // TMO instantiation
    tms          TMO_start_time = tm4_DCS_age (2 * 1000 * 1000);
    SampleTMO2   T1 (_T("MySecondTMO"), TMO_start_time);

    MainThrSleep ();
}
    
```

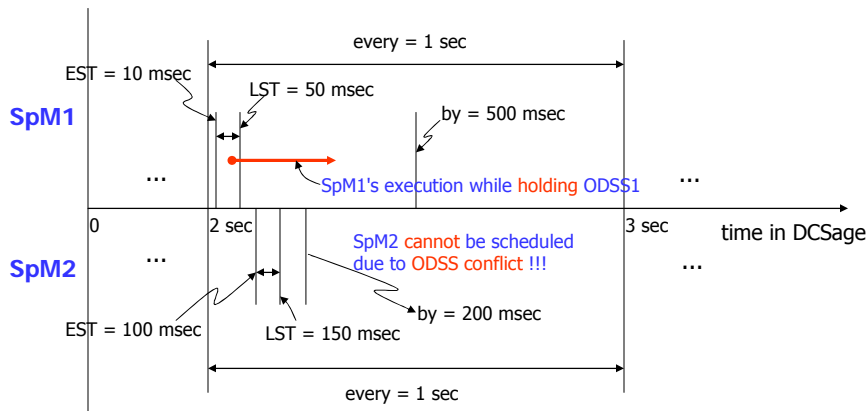
Feb-07 73

UCI  
DREAM Lab



## Sample Program VIII: TMO with ODSS Sharing (cont.)

- **Without** any early ReleaseODSS() call, SpM2 may miss its deadline due to ODSS conflict, though there is enough CPU time to execute both SpM methods concurrently.



Feb-07 74

UCI  
DREAM Lab



## Sample Program VIII: TMO with ODSS Sharing (cont.)

- **Without** early release of ODSS

```

void SampleTMO2::SpM1 ()
{
    TMOSLprintf ( _T(" [%d] This is SpM1!Wn"), ODSS1.Count);
    ODSS1.Count ++;
    // From now on, do more work without accessing ODSS1 any more.

    // Continue to work without releasing ODSS1.
    ...
}

void SampleTMO2::SpM2 ()
{
    TMOSLprintf ( _T(" [%d] This is SpM2!Wn"), ODSS1.Count);
    ODSS1.Count ++;
}
// end of SampleTMO2.cpp
    
```

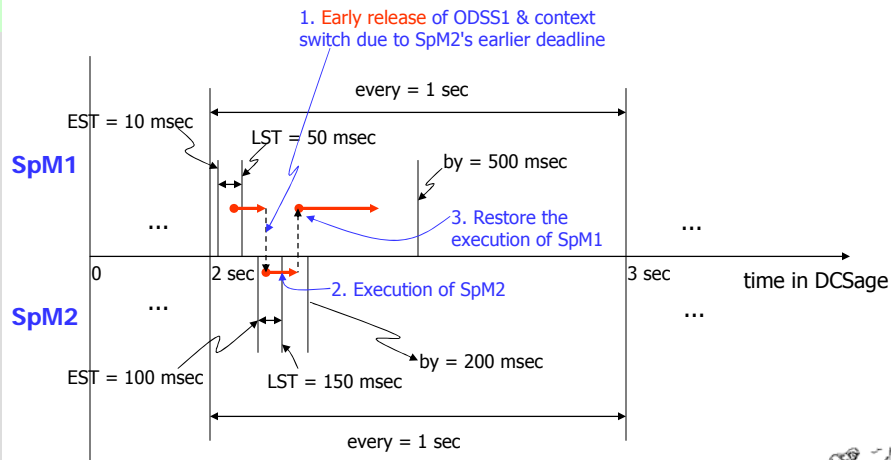
Feb-07 | 75

UCI  
DREAM Lab



## Sample Program VIII: TMO with ODSS Sharing (cont.)

- **With** an early ReleaseODSS() call, SpM2 can be scheduled on time.  
=> Provides **more concurrency !!!**



Feb-07 | 76

UCI  
DREAM Lab



## Sample Program VIII: TMO with ODSS Sharing (cont.)

- **With** early release of ODSS

```
void SampleTMO2::SpM1 ()
{
    TMOStprintf ( _T(" [%d] This is SpM1!\n"), ODSS1.Count);
    ODSS1.Count ++;
    // From now on, do more work without accessing ODSS1 any more.
    // So, release ODSS1 early!!!
    ODSS1.ReleaseODSS (); ← Release ODSS1 early so that SpM2
    // Continue to work after releasing ODSS1. can be scheduled and access ODSS1.
    ...
}

void SampleTMO2::SpM2 ()
{
    TMOStprintf ( _T(" [%d] This is SpM2!\n"), ODSS1.Count);
    ODSS1.Count ++;
}
// end of SampleTMO2.cpp
```

Feb-07

77

UCI  
DREAM Lab

## Ordered Isolation (OI) Rule

- Initiation timestamp (I-timestamp)
  - In the case of an SpM execution, the *I-timestamp* is defined as the record of the time instant at which the SpM execution was initiated according to the AAC specification of the SpM.
  - In the case of an SvM execution, the *I-timestamp* is defined as the record of the time instant at which the execution engine initiated the SvM execution after receiving the client request for the SvM execution and ensuring that the SvM execution can be initiated without violating the BCC and other execution rules.
- Ordered isolation rule
  - A method execution with an older I-timestamp must not be waiting for the release of an ODSS held by a method execution with a younger I-timestamp.
  - In addition, a method execution may never be rolled back due to an ODSS conflict.

Feb-07

78

UCI  
DREAM Lab

## ICT time-stamp

- The term "initiation time-stamp" was already introduced in the the previous slide.
- However, we need an additional time-stamp which is associated with each initiation candidate (i.e., method execution), can be used as a priority number, and leads to issuing such proper I-timestamps satisfying the OI rule.  
This additional time-stamp is called the [init-cand-ticket \(ICT\) time-stamp](#).
- The ICT-timestamp of an SpM is [LST](#) of the SpM.
- The ICT-timestamp of an SvM is the moment at which the SR is recognized by [VCT](#) of the server node.

Feb-07

79

UCI  
DREAM Lab

## Early Release of ODSS

- The designer of a TMO method may insert a call for "ReleaseODSS ()" in the function body of the TMO method where it is obvious that the associated ODSS will not accessed during the remainder of the method execution.
  - Such an early release will enable earlier initiation of some other TMO method executions than in the cases where such early releases are not exercised.
  - Once a TMO method execution releases an ODSS early via ReleaseODSS (), it cannot lock the ODSS again before it terminates.

Feb-07

80

UCI  
DREAM Lab

## Summary

---

- Essence of RT Programming:

At time T do S

{ = Start S during [T - Δ, T + Δ] }

- TMO, a high-level RT object, and TMO programming scheme
- TMOSL for application programming interface and TMO toolkit for application development
- TMO instantiation as the first step of TMO programming
- SpM, a time-triggered method of TMO
- ODSS, a unit of sharable data segment in each TMO

Feb-07	81
--------	----

UCI  
DREAM Lab



## References

---

- Kim, K.H., "APIs for Real-Time Distributed Object Programming," IEEE Computer, June 2000, pp.72-80.  
[http://dream.eng.uci.edu/TMO/pdf/computer\\_200006.pdf](http://dream.eng.uci.edu/TMO/pdf/computer_200006.pdf)
- TMO Support Library User Manual (draft) included in the TMO toolkit

Feb-07	82
--------	----

UCI  
DREAM Lab

