Examen Construction d'Applications Réparties

M1 - Master Sciences et Technologies
Mention Informatique

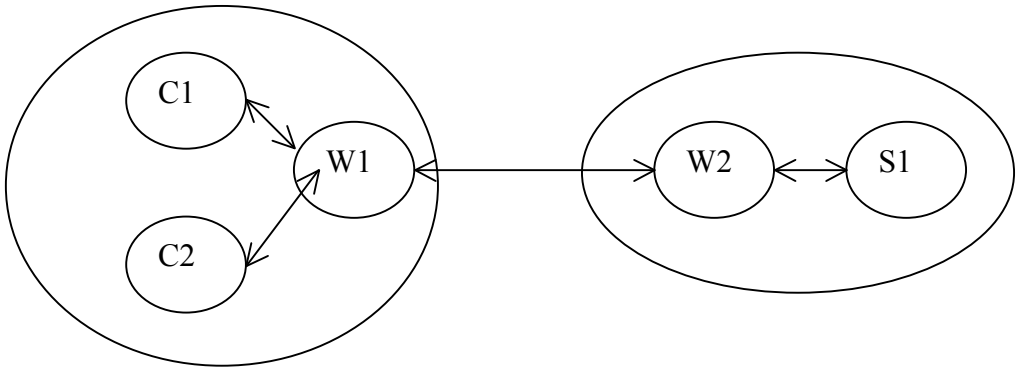
Université des Sciences et Technologies de Lille

2004-2005 – 2nde session

3 heures – Tous documents autorisés

1. Communication entre applications – CORBA et HTTP (10 points)

L'application répartie considérée est composée d'objets CORBA communiquant via des appels de méthodes. Le réseau sur lequel est déployée cette application est composé de deux sous-réseaux reliés par une passerelle de type coupe-feu qui permet uniquement le passage des paquets du protocole HTTP. Aussi, pour communiquer avec l'objet serveur S1, les objets clients C1 et C2 doivent passer par les serveurs Web W1 et W2.



Réseau

1.1 A votre avis, pour quelle raison a-t-on partitionné le réseau ? Quel est le seul port ouvert entre les deux sous-réseaux ? (1 point)

Serveur S1

L'objet serveur S1 fournit les méthodes suivantes :

- setTime : positionne l'heure courante et le fuseau horaire courant. Cette méthode prend en paramètres une chaîne de caractères représentant l'heure courante et un entier correspondant au fuseau horaire.

- getTime : retourne l'heure courante dans le fuseau horaire donné. Cette méthode prend en paramètre un entier correspondant au fuseau horaire et retourne une chaîne de caractères correspondant à l'heure courante dans le fuseau horaire transmis.

1.2 Ecrire en IDL CORBA l'interface ServeurItf de l'objet S1. (1 point)

Serveur Web W1

On s'intéresse au serveur Web W1 qui reçoit les requêtes des objets C1 et C2 et les transmet à W2.

1.3 Vis-à-vis de C1 et C2, W2 est-il un client Web, un client CORBA, un serveur Web ou un serveur CORBA ? Même question vis-à-vis de W2 ? (0,5 point)

Pour envoyer une invocation de méthode (setTime ou getTime) provenant de C1 ou de C2 à W2, W1 utilise une requête GET (http version 1.1). On suppose que l'adresse du serveur est w2.com. La classe W1Imlp fournit le code de W1.

1.4 Dans le cas de l'invocation setTime(« 10h15 »,1) indiquer qu'elle pourrait être la requête http envoyée par W1 ? Justifier votre réponse. (1 point)

1.5 Ecrire le code Java de la méthode setTime de la classe W1Impl. Pour simplifier on considère que la classe OutputStream possède une méthode write qui permet d'envoyer une chaîne de caractères. (1,5 point)

1.6 En quoi le code de la méthode getTime diffère de celui de setTime (on ne demande pas le code de la méthode, mais uniquement la différence) ? (0,5 point)

1.7 On se place dans le cas où C1 et C2 utilisent chacun plusieurs threads et invoquent simultanément les méthodes setTime et getTime de la classe W1Impl. Y-a-t-il un nouveau problème à résoudre ? (1 point)

Serveur Web W2

1.8 Ecrire en français l'algorithme du serveur W2 (1,5 point)

Ajout d'un serveur S2

On suppose qu'il y a maintenant un objet serveur RMI supplémentaire, S2, et que C1 et C2 peuvent choisir l'objet serveur (S1 ou S2) auquel ils veulent envoyer leur requête.

1.9 Les deux objets serveurs CORBA S1 et S2 peuvent-ils être des instances de la même classe ? Quelles sont les règles propres à CORBA que doit respecter une classe d'objet serveur ? (on ne demande pas d'écrire la classe) (1 point)

1.10 Pour C1, C2 et W1, est-ce que l'introduction de S2 change quelque chose ? Si non pourquoi ? Si oui pourquoi ? (1 point)

2. Mise en œuvre d'une application Web (7 points)

Une petite entreprise dispose sur la machine « staff » de la secrétaire d'une base de données permettant de gérer les congés des salariés. Dans le cadre de la mise en œuvre de l'intranet, le PDG de la entreprise souhaite offrir la possibilité à l'ensemble des employés de consulter le solde des jours de congés, de RTT, de la récupération disponibles et de poser des demandes de congés.

Pour se faire il pose le cahier des charges suivant. Il souhaite construire la nouvelle application à partir de l'existant qui est constitué de la base de données sur la machine de la secrétaire (base de données que l'on peut interroger en respectant le protocole htbd) et d'un serveur Web interne déployé sur la machine « admin». Il aimerait que vous développiez une application permettant à n'importe quel employé de consulter le solde des jours de congés et

de poser une demande de congés à partir de n'importe quel poste de l'entreprise possédant un navigateur Web.

2.1 Proposer une architecture permettant de répondre aux exigences du PDG. Vous identifierez sur cette architecture, les machines, les technologies utilisées sur ces machines (noms des outils), ainsi que les protocoles de communications entre ces machines. (1 point)

2.2 Sur cette architecture, vous identifierez pour chaque module logiciel son rôle (client, serveur) par rapport aux autres modules avec lesquels il interagit. (1 point)

2.3. Nous partons maintenant du principe que l'application que vous avez à développer est constituée de deux pages Web. La première permet à chaque employé de s'authentifier, la seconde lui permet de consulter le solde de congé et de poser une demande de jour de congés. Le protocole http étant un protocole en mode déconnecté, garantir l'authentification entre les deux pages Web n'est pas automatique. Proposer un mécanisme permettant de contrôler que le navigateur qui sert à l'authentification est le même que celui qui sert à poser de nouveaux jours de congés. (1 point)

2.4. Pour des questions de performances, il n'est pas idéal d'établir une nouvelle connexion à la base de données à chaque requête utilisateur.

- Comment, à l'aide des choix d'architecture que vous avez faits dans la question 1, vous pouvez éviter ce problème ? (0,5 point)
- A quel moment fermera-t-on la connexion ? (0,5 point)

2.5. L'administrateur système ajoute au cahier des charges la possibilité de déployer prochainement la base de données sur une autre machine. Aussi, il souhaite que les informations relatives à la connexion à la base de données (*nom de machine*, *numero de port, nom de la base de données*) ne se retrouvent pas noyées dans le code applicatif. Toujours en lien avec les choix technologiques effectués à la question 1, proposer un mécanisme permettant de stocker ces informations ailleurs. (1 point)

2.6. Nous nous concentrons maintenant sur le code qui va mettre en œuvre les mécanismes des questions 3, 4, 5, ainsi que permettre l'authentification à travers la première page Web et la génération d'une partie (contenant les informations sur l'utilisateur) de la deuxième page Web. Vous donnerez cette implémentation sous la forme de pseudo-code. (2 points)

Pour cette mise en oeuvre, vous avez à votre disposition entre autres les méthodes suivantes :

```
Public class ConnexionBD {

// Constructeur
public ConnexionBD getConnexionBD(String nomMachine,
                                  String portMachine,
                                  String nomBD) {…} ;
// Connexion à la base de données
public void connect() {…} ;

// Vérifie d'abord que l'utilisateur (login, passd) est bien connu dans la base de données.
// Si c'est le cas, un objet InfoUtil contenant toutes les informations de congés concernant
// l'utilisateur, est retourné.
public InfoUtil getInfoUser(String login, String passwd)  {…} ;
```

```
// Deconnexion à la base de données
public void deconnect() {…};

}

public class InfoUtil {

…

// A partir d'un objet infoUtil retourne une chaîne de caractères correspondant à du code html
// permettant la présentation des informations de d'un utilisateur
public String infoUtil2html(InfoUtil utilisateur) {…} ;

}
```

3. Sérialisation/Désérialisation (4 points)

3.1 Expliquer en quelques lignes le principe de la sérialisation et de la désérialisation (0,5 point). Quel est l'intérêt de la sérialisation et dans quel contexte doit-on utiliser un tel mécanisme ? (0,5 point)

3.2 La sérialisation et désérialisation en JAVA suivent des règles bien précises qui sont définies par le *Object Serialization Stream Protocol.* La spécification de ce protocole vous est fournie en annexe, ainsi qu'une table ascii.

A l'aide de ces documents, décoder le code sérialisé suivant:

```
ac ed 00 05 73 72 00 08  45 74 75 64 69 61 6e 74
99 7d 3c 17 6f 9d 44 e2  02 00 02 49 00 04 6e 6f
74 65 4c 00 03 6e 6f 6d  74 00 12 4c 6a 61 76 61
2f 6c 61 6e 67 2f 53 74  72 69 6e 67 3b 78 70 00
00 00 10 74 00 06 44 75  72 61 6e 74
```

Vous prendrez soin de bien commenter/expliquer chaque étape de votre décodage. (2 points)

3.3 Pour finir, vous donnerez la définition de la classe l'objet que vous avez décodé ainsi que la valeur des différents attributs. (1 point)

Annexe

Spécification du mécanisme d'encodage des données (*Object Serialization Stream Protocol*) du langage Java.

Les éléments du flot d'octets

Un minimum de structuration est nécessaire pour représenter les objets dans le flot. Chaque attribut de l'objet doit être présent : sa classe, les champs et les données. Ceux-ci pourront être lus par des méthodes spécifiques à la classe. La représentation des objets dans le flot est décrite par une grammaire. Une représentation particulière est prévue pour les objets nuls, les nouveaux objets, les classes, les tableaux, les chaînes de caractères et les références sur des

objets déjà dans le flot. Chaque objet écrit dans le flot est référencé de façon à pour voir être accessible. Les références commencent à 0.

Une classe d'objet est représentée par son objet ObjectStreamClass.

Un objet ObjectStreamClass est représenté par :
- Un SUID (Stream Unique Identifier) de classes compatibles,
- Un drapeau indiquant si la classe a des méthodes de lecture/écriture des objets,
- Le nombre de champs non statiques ou non temporaires (*transients*),
- Le tableau de champs de la classe qui est sérialisé par le mécanisme par défaut. Pour les tableaux et les champs de l'objet, le type du champ est inclus comme une chaîne de caractères.
- Les enregistrements de blocs de données ou les objets optionnels écrits par une méthode annotateClass.
- L'ObjectStreamClass de son super-type (null si la superclasse n'est pas sérialisable).

Les chaînes sont représentées par leur encodage UTF (Unified Text Format).

Les tableaux sont représentés par
- Leur objet ObjectStreamClass
- Le nombre d'éléments
- La séquence de valeurs. Le type des valeurs est implicite par le type du tableau. Par exemple les valeurs d'un tableau d'octets sont de type octet

Les nouveaux objets dans le flot sont représentés par :

- La classe la plus dérivée de l'objet
- Les données pour chaque classe sérialisable de l'objet, avec la superclasse la plus haute en premier. Pour chaque classe, le flux contient :
    - Les champs sérialisés par défaut (les champs ni statiques ni temporaires comme décrits dans la ObjectStreamClass correspondante)
    - Si la classe a des méthodes writeObject/readObject, il peut y avoir des objets ou des blocs de données optionnels des types de primitives écrits par la méthode writeObject suivi par un code endBlockData.

La grammaire du format flot d'octet

Nous suivons les règles typographiques suivantes. Les symboles non terminaux sont en italique, les symboles terminaux sont dans en majuscule et en gras. Les définitions de non-terminaux sont suivis d'un « . ». La définition est suivie par une ou plusieurs alternatives, chacune sur une ligne séparée

La table suivante décrit la notation :

Notation	Significations
(*datatype*)	Ce token a le type de données spécifié, par exemple byte
token[n]	Un nombre prédéfini d'occurrences du token qui est un tableau
x0001	Une valeur exprimée en hexadécimal. Le nombre de digit hexa reflète la taille de la valeur
<xxx>	Une valeur lue à partir du flot utilisé pour indiquer la longueur d'un tableau

Un flot d'octet est représenté par n'importe quel flot satisfaisant la règle *stream*

```
stream:
  magic version contents

[1]contents:
  content
  contents content

[2]content:
  object
  blockdata

[3]object:
  newObject
  newClass
  newArray
  newString
  newClassDesc
  prevObject
  nullReference
  exception
  TC_RESET

[4]newClass:
  TC_CLASS classDesc newHandle

[5]classDesc:
  newClassDesc
  nullReference
  (ClassDesc)prevObject      // an object required to be of type
                             // ClassDesc

[6]superClassDesc:
  classDesc

[7]newClassDesc:
  TC_CLASSDESC className serialVersionUID newHandle classDescInfo
  TC_PROXYCLASSDESC newHandle proxyClassDescInfo

[8]classDescInfo:
  classDescFlags fields classAnnotation superClassDesc

[9]className:
  (utf)

[10]serialVersionUID:
  (long)

[11]classDescFlags:
  (byte)                  // Defined in Terminal Symbols and
                            // Constants

[12]proxyClassDescInfo:
  (int)<count> proxyInterfaceName[count] classAnnotation
      superClassDesc

[13]proxyInterfaceName:
  (utf)

[14]fields:
```

```
  (short)<count>  fieldDesc[count]

[15]fieldDesc:
  primitiveDesc
  objectDesc

[16]primitiveDesc:
  prim_typecode fieldName

[17]objectDesc:
  obj_typecode fieldName className1

[18]fieldName:
  (utf)

[19]className1:
  (String)object             // String containing the field's type,
                             // in field descriptor format

[20]classAnnotation:
  endBlockData
  contents endBlockData      // contents written by annotateClass

[21]prim_typecode:
  `B'   // byte
  `C'   // char
  `D'   // double
  `F'   // float
  `I'   // integer
  `J'   // long
  `S'   // short
  `Z'   // boolean

[22]obj_typecode:
  `['   // array
  `L'   // object

[23]newArray:
  TC_ARRAY classDesc newHandle (int)<size> values[size]

[24]newObject:
  TC_OBJECT classDesc newHandle classdata[]  // data for each class

[25]classdata:
  nowrclass                 // SC_SERIALIZABLE & classDescFlag &&
                            // !(SC_WRITE_METHOD & classDescFlags)
  wrclass objectAnnotation  // SC_SERIALIZABLE & classDescFlag &&
                            // SC_WRITE_METHOD & classDescFlags
  externalContents          // SC_EXTERNALIZABLE & classDescFlag &&
                            // !(SC_BLOCKDATA  & classDescFlags
  objectAnnotation          // SC_EXTERNALIZABLE & classDescFlag&&
                            // SC_BLOCKDATA & classDescFlags

[26]nowrclass:
  values                    // fields in order of class descriptor

[27]wrclass:
  nowrclass

[28]objectAnnotation:
```

```
  endBlockData
  contents endBlockData     // contents written by writeObject
                            // or writeExternal PROTOCOL_VERSION_2.

[29]blockdata:
  blockdatashort
  blockdatalong

[30]blockdatashort:
  TC_BLOCKDATA (unsigned byte)<size> (byte)[size]

[31]blockdatalong:
  TC_BLOCKDATALONG (int)<size> (byte)[size]

[32]endBlockData    :
  TC_ENDBLOCKDATA

[33]externalContent:          // Only parseable by readExternal
  ( bytes)                 // primitive data
    object

[34]externalContents:         // externalContent written by
  externalContent         // writeExternal in PROTOCOL_VERSION_1.
  externalContents externalContent

[35]newString:
  TC_STRING newHandle (utf)
  TC_LONGSTRING newHandle (long-utf)

[36]prevObject
  TC_REFERENCE (int)handle

[37]nullReference
  TC_NULL

[38]exception:
  TC_EXCEPTION reset (Throwable)object   reset

[39]magic:
  STREAM_MAGIC

[40]version
  STREAM_VERSION

[41]values:          // The size and types are described by the
                  // classDesc for the current object

[42]newHandle:       // The next number in sequence is assigned
                  // to the object being serialized or deserialized

[43]reset:           // The set of known objects is discarded
                  // so the objects of the exception do not
                  // overlap with the previously sent objects
                  // or with objects that may be sent after
                  // the exception
```

Les symboles et les constantes terminaux

Les symboles suivants dans java.io.ObjectStreamConstants définissent les valeurs terminales et les valeurs des constantes attendues dans un flot :

```
final static short STREAM_MAGIC = (short)0xaced;
final static short STREAM_VERSION = 5;
final static byte TC_NULL = (byte)0x70;
final static byte TC_REFERENCE = (byte)0x71;
final static byte TC_CLASSDESC = (byte)0x72;
final static byte TC_OBJECT = (byte)0x73;
final static byte TC_STRING = (byte)0x74;
final static byte TC_ARRAY = (byte)0x75;
final static byte TC_CLASS = (byte)0x76;
final static byte TC_BLOCKDATA = (byte)0x77;
final static byte TC_ENDBLOCKDATA = (byte)0x78;
final static byte TC_RESET = (byte)0x79;
final static byte TC_BLOCKDATALONG = (byte)0x7A;
final static byte TC_EXCEPTION = (byte)0x7B;
final static byte TC_LONGSTRING = (byte) 0x7C;
final static byte TC_PROXYCLASSDESC = (byte) 0x7D;
final static  int   baseWireHandle = 0x7E0000;
```

L'octet flag classDescFlags peut inclure les valeurs suivantes :

```
final static byte SC_WRITE_METHOD = 0x01; //if SC_SERIALIZABLE
final static byte SC_BLOCK_DATA = 0x08;    //if SC_EXTERNALIZABLE
final static byte SC_SERIALIZABLE = 0x02;
final static byte SC_EXTERNALIZABLE = 0x04;
```

The flag `SC_WRITE_METHOD` is set if the Serializable class writing the stream had a `writeObject` method that may have written additional data to the stream. In this case a `TC_ENDBLOCKDATA` marker is always expected to terminate the data for that class.

The flag `SC_BLOCKDATA` is set if the `Externalizable` class is written into the stream using `STREAM_PROTOCOL_2`. By default, this is the protocol used to write `Externalizable` objects into the stream in JDK™ 1.2. JDK™ 1.1 writes `STREAM_PROTOCOL_1`.

The flag `SC_SERIALIZABLE` is set if the class that wrote the stream extended `java.io.Serializable` but not `java.io.Externalizable`, the class reading the stream must also extend `java.io.Serializable` and the default serialization mechanism is to be used.

The flag `SC_EXTERNALIZABLE` is set if the class that wrote the stream extended `java.io.Externalizable`, the class reading the data must also extend `Externalizable` and the data will be read using its `writeExternal` and `readExternal` methods.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.asciitable.com