Appendix B

SOPA: Technical Details

This section presents technical insights into language grammar and structures of the SOPA framework in addition to the description in Chapter 4.

B.1 DTD of SOPA Graph Serialization

The following listing presents the DTD of SOPA's default media graph description.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE SOPA:script [
        <!ENTITY source "de.echalk.sopa.SourceNode">
        <!ENTITY target "de.echalk.sopa.TargetNode">
        <!ENTITY pipe "de.echalk.sopa.PipeNode">
        <!ENTITY mixer "de.echalk.sopa.MixerNode">
        <!ENTITY fork "de.echalk.sopa.ForkNode">
        <!ENTITY generic "de.echalk.sopa.MediaNode">
        <!ELEMENT SOPA:script ((synchronize*)|(on*)|(include*)|(loadprops*)
                        |(setproperty*)|(service*))>
        <!ELEMENT on ((synchronize*)|(on*)|(include*)|(loadprops*)
                        |(setproperty*)|(service*))>
                <!ATTLIST on match CDATA #REQUIRED>
        <!ELEMENT setproperty EMPTY>
                <!ATTLIST setproperty key CDATA #REQUIRED>
                <!ATTLIST setproperty value CDATA "">
                <!ATTLIST setproperty persistent (true|false) "false">
        <!ELEMENT error ((PCDATA)|(property))>
                <!ATTLIST error level (message|debug|warning|error|fatalerror) "error">
        <!ELEMENT include ((PCDATA)|(property))>
        <!ELEMENT loadprops ((PCDATA)|(property))>
        <!ELEMENT property EMPTY>
                <!ATTLIST property key CDATA #REQUIRED>
                <!ATTLIST property default CDATA "">
        <!ELEMENT service (((PCDATA)|(property*))*)>
                <!ATTLIST service label CDATA #REQUIRED>
                <!ATTLIST service type CDATA #REQUIRED>
                <!ATTLIST service match CDATA #REQUIRED>
                <!ATTLIST service target CDATA "">
        <!ELEMENT synchronize (((PCDATA)|(property*))*)>
]>
```

B.2 LDAP Query Syntax

SOPA's main mechanism to locate nodes and evaluate conditions are LDAP queries as standardized by RFC 1960. An impression of the power and the

limits of this language can be obtained from its syntax. The following BNF grammar has been verbosely extracted from RFC 1960 [Howes, 1996].

```
<filter> ::= '(' <filtercomp> ')'
     <filtercomp> ::= <and> | <or> | <not> | <item>
     <and> ::= '&' <filterlist>
     <or> ::= '|' <filterlist>
     <not> ::= '!' <filter>
     <filterlist> ::= <filter> | <filter> <filterlist>
     <item> ::= <simple> | <present> | <substring>
     <simple> ::= <attr> <filtertype> <value>
     <filtertype> ::= <equal> | <approx> | <greater> | <less>
     <equal> ::= '='
     <approx> ::= '~='
     <greater> ::= '>='
     <less> ::= '<='
     <present> ::= <attr> '=*'
     <substring> ::= <attr> '=' <initial> <any> <final>
     <initial> ::= NULL | <value>
     <any> ::= '*' <starval>
     <starval> ::= NULL | <value> '*' <starval>
     <final> ::= NULL | <value>
```

⟨`attr`⟩ is a string representing an AttributeType. ⟨`value`⟩ is a string representing an AttributeValue. If a ⟨`value`⟩ must contain one of the characters '*' or '(' or ')', these characters should be escaped by preceding them with the backslash character.

B.3 SOPA Command Line Commands

SOPA's command-line console allows direct access to some framework methods and provides debugging functionality. The following is a lists of all available commands as it is printed out on the console.

```
Basic commands:
help                        - Show this help
hide                        - Hide console
clear                       - Clear console
ver                         - Show application version
gc                          - Request garbage collection
finalize                    - Request finalization of pending objects
free                        - Show memory statistics
gsp                         - Get list of system properties
gp                          - Get list of all SOPA properties
spp <key> <value>           - Set persistent property <key> to <value>
swp <key> <value>           - Set work property <key> to <value>
rp <key>                    - Remove property named <key>
ts                          - List all threads
suspend <no>                - Suspend thread <no> (Caution: Read Java doc!)
resume <no>                 - Resume thread <no>  (Caution: Read Java doc!)
interrupt <no>              - Interrupt thread <no>
tstop <no>                  - Stop thread <no>    (Caution: Read Java doc!)
destroy <no>                - Destroy thread <no>
setPriority <no> <pri>      - Set priority of thread <no> to <pri>
shutdown                    - Request regular shutdown
seppuko                     - Immediate exit w/o saving (DANGEROUS: ALL UNSAVED DATA LOST)
save <filename>             - Dump console content to <filename>
publish <classname> <url>   - Publish node into Eureka network (if service available).
unpublish <classname> <url> - Remove node from Eureka network (if service available).
oscar <command>             - Pass <command> to oscar. (In case of ambiguity)

Commands for the Oscar subsystem:
bundlelevel <level> <id> ... | <id> - set or get bundle start level.
cd [<base-URL>]                     - change or display base URL.
headers [<id> ...]                  - display bundle header properties.
help                                - display shell commands.
install <URL> [<URL> ...]           - install bundle(s).
obr help                            - Oscar bundle repository.
```

```
packages [<id> ...]                - list exported packages.
ps [-l]                            - list installed bundles.
refresh                            - refresh packages.
services [-u] [-a] [<id> ...]      - list registered or used services.
shutdown                           - shutdown Oscar.
start <id> [<id> <URL> ...]        - start bundle(s).
startlevel [<level>]               - get or set framework start level.
stop <id> [<id> ...]               - stop bundle(s).
uninstall <id> [<id> ...]          - uninstall bundle(s).
update <id> [<URL>]                - update bundle.
version                            - display version of Oscar.
```

B.4 A Minimal PipeNode

The following code sample is a demonstration implementation of a minimal PipeNode that implements the identity function. The example shows that very few methods have to be implemented and a minimum set of propietary concepts are to be learned by any developer wanting to create a costum filter in the SOPA framework. Even though a PipeNode requires the methods of both a SourceNode and a TargetNode, no more than ten methods have to be implemented.

```
import java.util.*;
import de.echalk.sopa.*;

public class IdentityPipe extends PipeNode
{
    private FormatDescriptor outfd;

    // Metadata for graph resolution
    public int getVersion()
    {
        return 100; // is divided by 100 = 1.00.
    }

    public String getNodeName()
    {
        return "IdentityPipe";
    }

    public String getCopyright()
    {
        return getNodeName()+" (c) 2003-2006 by G. Friedland";
    }

    public Properties getProperties()
    {
        Properties p=new Properties();
        p.put("author","Friedland");
        // Put additional, optional properties here
        return p;
    }

    public FormatDescriptor[] needsAsInput()
    {
        return null; // null means: handles any format
    }

    public FormatDescriptor[] givesAsOutput(FormatDescriptor inputformat)
    {
        return new FormatDescriptor[]{inputformat};
    }

    // Operational code
    public void startWork(FormatDescriptor formattoproduce, TargetNode t)
    {
    }
```

```
    public int syncputdata(SourceNode sn, Object o, int numframes)
    {
        outfd=sn.getDataFormat(); // sn, o are guaranteed to be non-null
        return res=getTarget().putdata(this,o,numframes);
    }

    public void stopWork()
    {
    }

    public FormatDescriptor getDataFormat()
    {
        return outfd;
    }
}
```