École nationale supérieure des mines de Nantes
ASCOLA Research Group

AccLab User Guide

Version 1.0

June 5, 2015

Walid BENGHABRIT

Supervisor : **Prof. Jean-Claude Royer**
Co-Supervisor : **A/Prof. Hervé Grall**

Contents

Table of contents 1
1 Getting started 2
1.1 Install AccLab 2
1.2 Using AAL compiler "aalc" 2
1.3 Writing your first AAL program 2
1.4 Running the program 4
1.5 Using core libraries 5
1.6 Advanced checks 6
1.7 Using the shell 7
2 AAL language 10
3 Using AccLab web GUI 11

1 Getting started

1.1 Install AccLab

To use the ltl prover, you need to put the following executable files (tspass, fotl-translate) in `tools/your_platform/ (linux/mac/win)` TSPASS binaries are provided for linux x64 and mac x64 in the folder `tools/_platformeName_/`. For other platformes you have to compile tspass source code. The last version of TSPASS can be found in : <http://lat.inf.tu-dresden.de/ michel/software/tspass/>. The source code for TSPASS version 0.95-0.17 is provided with this tool. AAL Syntax highlighting modes for emacs, intellij, nano and ace, can be found in `tools/utils/`. If you want to run aalc using a symbolic link you need to set the environment variable `ACCLAB_PATH` : `export ACCLAB_PATH=<AccLab_install_dir>`. You need python3.4.0 or greater.

1.2 Using AAL compiler "aalc"

Listing 1: aalc options

```
root@root/:$ python aalc
aalc
  -h    --help                  display this help and exit
  -i    --input                 the input file
  -i    --output                the output file
  -c    --compile               compile the file, that can be loaded after using -l
  -m    --monodic               apply monodic check on aal file
  -s    --shell                 run a shell after handling aal program
  -k    --check                 perform a verbose check
  -l    --load                  load a compiled aal file (.aalc) and run a shell
  -t    --fotl                  translate the aal program into FOTL
  -r    --reparse               reparse tspass file
  -r    --recompile             recompile the external files
  -b    --no-colors             disable colors in output
  -x    --compile-stdlib        compile the standard library
  -d    --hotswap               enable hotswaping (for development only)
  -a    --ast                   show ast tree
  -u    --gui                   run the gui (BETA)
```


1.3 Writing your first AAL program

Let consider the following senario, we have three actors :

- cloud storage service : let call it `css` which is a cloud service provider
- alice and bob : an end users that uses css service

The `css` offers the following services : read (a user reads some data form css server), store (a user stores some data into css server), delete (a user deletes some data from css server). `css` allows users to read/store/delete only their data on his server, and don't allow them to read other customers data. `css` can also read and delete any data from his server.

Alice want to check if `css` policy respect her privacy. Typically she want to know if she is allowed to performs some actions and if bob can read here data.

a. **Declaring actors** : first we need to declare our actors

```
// Agents declaration
AGENT alice
AGENT bob
AGENT css
```

b. **Declaring services** the services used are :

```
SERVICE read
SERVICE store
SERVICE delete
```

c. **Linking services and actors**

```
AGENT alice TYPES(Actor) REQUIRED(read store delete) PROVIDED()
AGENT bob   TYPES(Actor) REQUIRED(read store delete) PROVIDED()
AGENT css   TYPES(Actor) REQUIRED() PROVIDED(read store delete)
```

d. **Defining policies**

```
/*
 * Cloud storage service provider policy
 */
CLAUSE css_policy (
    FORALL d:data FORALL a:Actor

    // Allow users to read their data
    IF (d.subject == a) THEN {
        PERMIT a.read[css](d)
    } AND

    // Deny access to read other
    IF (d.subject != a) THEN {
        DENY a.read[css](d)
    } AND

    // Allow css to read/delete stored data
    PERMIT css.read[css](d) AND
    PERMIT css.delete[css](d)
)

/*
 * Alice's preferences
 */
CLAUSE alice_policy (
    FORALL d:data
    // Alice want to be able to read all her data stored on css
    IF (d.subject == alice) THEN {
        PERMIT alice.read[css](d)
    }
)
```

e. **Writing checks** Now we want to check if `Alice`'s privacy preferences are respected by the `css` policy. To do this, we can call the macro `validate` and passing the the clauses names as arguments. Important : Note that the order of arguments is important.

```
CALL validate("css_policy" "alice_pref")
```

1.4 Running the program

- Run the AAL program

```
root@root/:$ python aalc -i examples/tuto0.aal

------------------------- Monodic check -------------------------
Monodic check passed !
------------------------- Starting Validity check -------------------------
c1 : css_policy
c2 : alice_pref
----- Checking c1 & c2 consistency :
  -> Satisfiable
----- Checking c1 => c2 :
  -> Satisfiable
----- Checking ~(c1 => c2) :
  -> Unsatisfiable

[VALIDITY] Formula is valid !
------------------------- Validity check End -------------------------

File : examples/tuto0.aal

Execution time : 0.24277639389038086
```

Here the result of

- Perform an detailed check

```
root@root/:$ python aalc -i examples/tuto0.aal -k

------------------------- Start Checking -------------------------

** DECLARATIONS
[DECLARED AGENTS]   : 3
[DECLARED SERVICES] : 6
[DECLARED DATA]     : 0
[DECLARED TYPES]    : 10

*** Forwards references check
[AGENTS]   : 0
[SERVICES] : 0
[DATA]     : 0
[TYPES]    : 0

** LOADED libraries
[LIBS] : 2

** CLAUSES
[CLAUSES] : 2
Monodic test :
|css_policy | Formula is monodic ! |
|alice_pref | Formula is monodic ! |
------------------------- Checking End -------------------------
```

- Perform monodic test on all clauses :

```
root@root/:$ python aalc -i examples/tuto0.aal -m

------------------------- Start Checking -------------------------
|css_policy | Formula is monodic ! |
```

```
|alice_pref | Formula is monodic ! |
------------------------- Checking End ------------------------
```

- Translate AAL program into FOTL (in tspass syntax):

```
root@root/:$ python aalc -i examples/tuto0.aal -t

------------------------ FOTL Translation start ------------------------

%%%%%%%%% START EVN %%%%%%%%%%
%%% Types knowledge
data(CTS)  &
Actor(CTS)  &
DataSubject(CTS)  &
DataController(CTS)  &
DataProcessor(CTS)  &
DwDataController(CTS)  &
Auditor(CTS)  &
CloudProvider(CTS)  &
CloudCustomer(CTS)  &
EndUser(CTS)  &


%%% Action authorizations
(![x, y, z] (read(x, y, z) => Pread(x, y, z))) &
(![x, y, z] (store(x, y, z) => Pstore(x, y, z))) &
(![x, y, z] (delete(x, y, z) => Pdelete(x, y, z))) &
(![x, y, z] (read(x, y, z) => Pread(x, y, z))) &
(![x, y, z] (write(x, y, z) => Pwrite(x, y, z))) &
(![x, y, z] (audit(x, y, z) => Paudit(x, y, z))) &


%%% Actors knowledge
Actor(alice)  &
Actor(bob)  &
Actor(css)  &

%%%%%%%%% END EVN %%%%%%%%%%

%% Clause : css_policy
always(![d] ( data(d) & (![a] ( Actor(a) & (((( (subject(d, a) => Pread(a, css, d)) &
(~subject(d, a) => ~Pread(a, css, d))) & Pread(css, css, d)) & Pdelete(css, css, d)))))) )
%% Clause : alice_pref
always(![d] ( data(d) & ( (subject(d, alice) => Pread(alice, css, d)))) )


------------------------- FOTL Translation end -------------------------
```


1.5 Using core libraries

You can load external AAL files using `LOAD "aal_file"`(without the extension)

core.types Contains the basic types declarations (Actor, DataSubject, DataController, DataProcessor, ...)

```
LOAD "core.types"
```

core.macros Contains some useful macros.

```
// Loading lib
LOAD "core.macros"
```

1.6 Advanced checks

```
/*
 * Alice's preferences
 */
CLAUSE alice_policy (
    FORALL d:data
    // Alice want to be able to read all her data stored on css
    IF (d.subject == alice) THEN {
        PERMIT alice.read[css](d)

        // Bob cannot read Alice's data
        DENY bob.read[css](d)
    }
)
```

The previous call to validate macro will gives : Satisfiable.

Why ? Because the predicate subject is not exclusive : subject of d can be alice and bob at the same time.

A simple way to fix it is to add the condition that the subject of d is not bob :

```
IF (d.subject == alice AND d.subject != bob) THEN {
....
```

Or we can to add the following condition manually to our check :

```
(![f] (subject(f, alice) => ~subject(f, bob))) &
```

The construction CHECK allows you to write directly FOTL formula mixed with somme AAL constructions.

- @verbose (print the generated formula)
- @buildenv (build the environment, which is a set of preconditions generated from the AAL program)
- clause(c) : get the fotl translation of clause "c"
- clause(c).ue : get the fotl translation of usage part of the clause "c"
- clause(c).ae : get the fotl translation of audit part of the clause "c"
- clause(c).re : get the fotl translation of rectification part of the clause "c"
- APPLY chk() : call the check "chk"

```
CHECK c1() {

@verbose
~(
    % Build the environment
```

```
    @buildenv

    % Add extra condition
    (![f] (subject(f, alice) => ~subject(f, bob))) &

    % The check P => U
    (clause(css_policy).ue)
    =>
    (clause(alice_pref).ue)
)

}
APPLY c1()
```

The result is Unsatisfiable so the formula is valide.

1.7 Using the shell

The shell is a useful tool for developing :

- Run the shell.

```
root@root/:$ python aalc -i examples/tuto0.aal -s
....
shell >
```

- Type help to show the shell help.

```
shell >help
  Shell Help
 - call(macro, args)     call a macro where /
                          *macro : is the name of the macro
                          *args : a list of string; <<ex :["'args1'", "'args2'", ..."'argsN'"]>>
 - clauses()             show all declared clauses in the loaded aal program
 - macros()              show all declared macros in the loaded aal program
 - load(lib)             load the librarie lib
 - quit / q              exit the shell
 - help / h / man()      show this help
 - self                  the current compiler instance of the loaded aal program
 - aalprog               the current loaded aal program
 - man(arg)              print the help for the given arg
 - hs(module)            hotswaping : reload the module
 - r()                   hot-swaping the shell
```

- Here an example, we print all clauses in the AAL program.

```
shell> clauses()
css_policy alice_pref
```

- self variable refers to the compiler instance.

```
shell> self
<AALCompiler.AALCompilerListener object at 0x7f8b00ce8630>
```

- man can be called on any element, it will show its documentation.

```
shell> man(self)
printing manual for <class 'AALCompiler.AALCompilerListener'>
Manual for aal compiler visitor
 - Attributes
        -  aalprog      Get the AAL program instance
        -  file         The AAL source file
        -  libs         Show the loaded libraries
        -  libsPath     Print the standard lib path
 - Methods
        - load_lib(lib_name)    Load an aal file
        - clause(clauseId)      Lookup for clause cluaseId
        - show_clauses()        Show all clauses (names
        - get_clauses()         Get all clauses (objects)
        - get_macros()          Get all macros (objects)
```

- AAL program instance

```
shell> man(aalprog)
printing manual for <class 'AALMetaModel.m_aalprog'>

    AAL program class.
    Note that clauses and macros extends a declarable type, but are not in the declarations dict

    Attributes
        - clauses: a list that contains all program clauses
        - declarations: a dictionary that contains lists of typed declarations
        - comments: a list that contains program s comment
        - macros: a list that contains program s macros declarations
        - macroCalls: a list that contains program s comment
```

- Calling a macro

```
call("validate", ["'css_policy'", "'alice_pref'"])
------------------------- Monodic check ------------------------
Monodic check passed !
------------------------- Starting Validity check -------------------------
c1 : css_policy
c2 : alice_pref
----- Checking c1 & c2 consistency :
  -> Satisfiable
----- Checking c1 => c2 :
  -> Satisfiable
----- Checking ~(c1 => c2) :
  -> Satisfiable
:: Solving trigger

[VALIDITY] Formula is not valid !
------------------------- Validity check End -------------------------
```

- Defining a new macro

```
shell> self.new_macro("toto", ["p"], """print(p)""")
shell> call("toto", ["'4'"])
4
```

- hotswaping commands are used for debugging purpose only. r() command allows you to reload the shell without exiting it after source code modification.

- hs(module) reloading other modules after source code modification without exiting. ! IMORTANT : to use hotswaping properly you must enable it explicitly in aalc arguments `-d / -hotswap`,

2 AAL language

Listing 2: AAL Syntax

```
// AAL CORE
AALprogram    ::= (Declaration | Clause | Comment | Macro | MacroCall | Loadlib
                  | LtlCheck | CheckApply | Exec | Behavior)
Declaration   ::= AgentDec | ServiceDec | DataDec | TypesDec | varDec
AgentDec      ::= AGENT Id [TYPES '(' Type *')' REQUIRED'('service*')' PROVIDED'('service*')']
ServiceDec    ::= SERVICE Id [TYPES '(' Type* ')'] [PURPOSE '(' Id* ')']
DataDec       ::= DATA Id TYPES '(' Type* ')' [REQUIRED'('service*')' PROVIDED'('service*')']
 SUBJECT agent
VarDec        ::= Type_Id Id [attr_Id '(' value* ')']*
Clause        ::= CLAUSE Id '(' [Usage] [Audit Rectification] ')'
Usage         ::= ActionExp
Audit         ::= AUDITING Usage
Rectification ::= IF_VIOLATED_THEN Usage
ActionExp     ::= Action | NOT ActionExp | Modality ActionExp | Condition
                  | ActionExp (AND|OR|ONLYWHEN|UNTIL|UNLESS) ActionExp
                  | Author | Quant* | IF '(' ActionExp ')' THEN '{' ActionExp '}'
Exp           ::= Variable | Constant | Variable.Attribute
Condition     ::= [NOT] Exp | Exp ['==' | '!='] Exp | Condition (AND|OR) Condition
Author        ::= (PERMIT | DENY) Action
Action        ::= agent.service ['['[agent]']'] '('Exp')' [Time] [Purpose]
Quant         ::= (FORALL | EXISTS) Var [WHERE Condition]
Variable      ::= Var ':' Type
Modality      ::= MUST | MUSTNOT | ALWAYS | NEVER | SOMETIME
Time          ::= (AFTER | BEFORE) Date | Time (AND | OR) Time
Date          ::= STRING
Type, var, val, attr Id, agent, Constant, Purpose ::= literal

// AAL Type extension
TypesDec      ::= TYPE Id [EXTENDS '(' Type* ')'] ATTRIBUTES '(' AttributeDec* ')' ACTIONS '('
 ActionDec* ')'
AttributeDec  ::= Id ':' Type
ActionDec     ::= Id
Type, Id      ::= litteral

// Reflexion extension
Macro         ::= MACRO Id '(' param* ')' '(' mcode ')'
MCode         ::= '"""' Meta model api + Python3 code (subset) '"""'
MCall         ::= CALL Id '(' param* ')'
LoadLib       ::= LOAD STRING;
Exec          ::= EXEC MCode

// FOTL checking extension
LtlCheck     ::= CHECK Id '(' param* ')' '(' check ')'
check        ::= FOTL_formula + clause(Id) [.ue | .ae | .re]
CheckApply   ::= APPLY Id '(' param* ')'

// Behavior extension
Behavior    ::= BEHAVIOR Id '(' ActionExp ')'
```

TODO : details

3 Using AccLab web UI

To run the web ui :

```
root@root/:$ python aalc.py -u
```

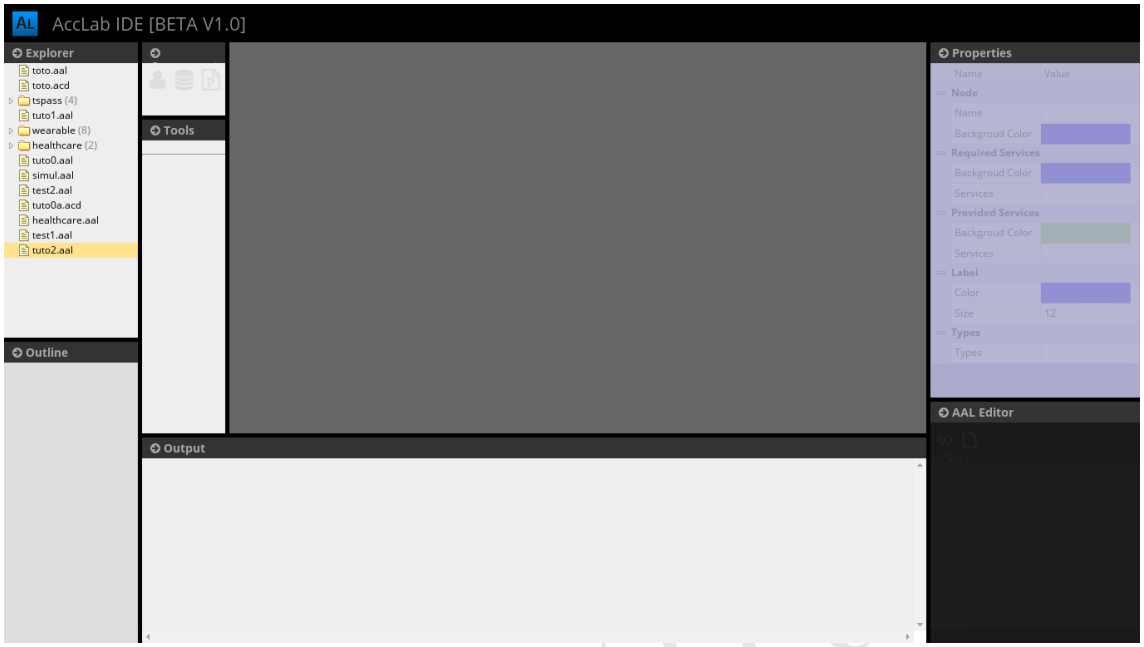

Figure 1: ACD Editor

1. **Explorer :** This panel contains a tree view of the your workspace (`-*.aal` a simple text format that contains AAL code; `-*.acd` a json file format that contains the components diagram)
2. **Outline :** The outline contains a tree view of the current components (with blue icon) in the opened `acd` file, and for each agent its required services (in red icon) and provided services (green icon).
3. **Components :** Contains the elements that can be used in the diagram (`agent` : a simple agent with types, required and provided services; `data` : same as agent but with an additional attribute `subject` (the data owner) ; macro : insert a macro call in the generated AAL program (more details will be provided soon)).
4. **Tools :** A panel containing different tools to be used while editing acd/aal files (copy-/past/zoom/save/etc).
5. **Diagram :** The diagram.
6. **Properties :** A grid containing the properties of a selected element in the diagram, in which you can edit its name, style properties (color, font, etc) and also types/services.

7. **Output :** The output where we show the result coming from the back-end.
8. **AAL editor :** Allows to edit quickly a component's policy, before generating the AAL program.

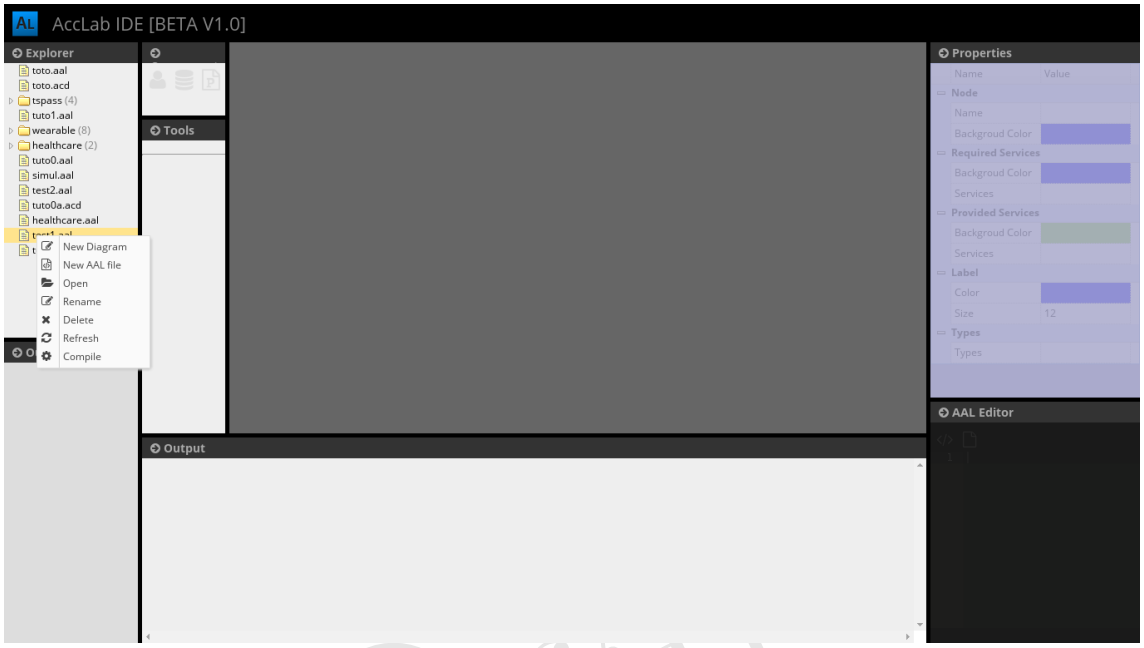

Figure 2: Creating new diagram

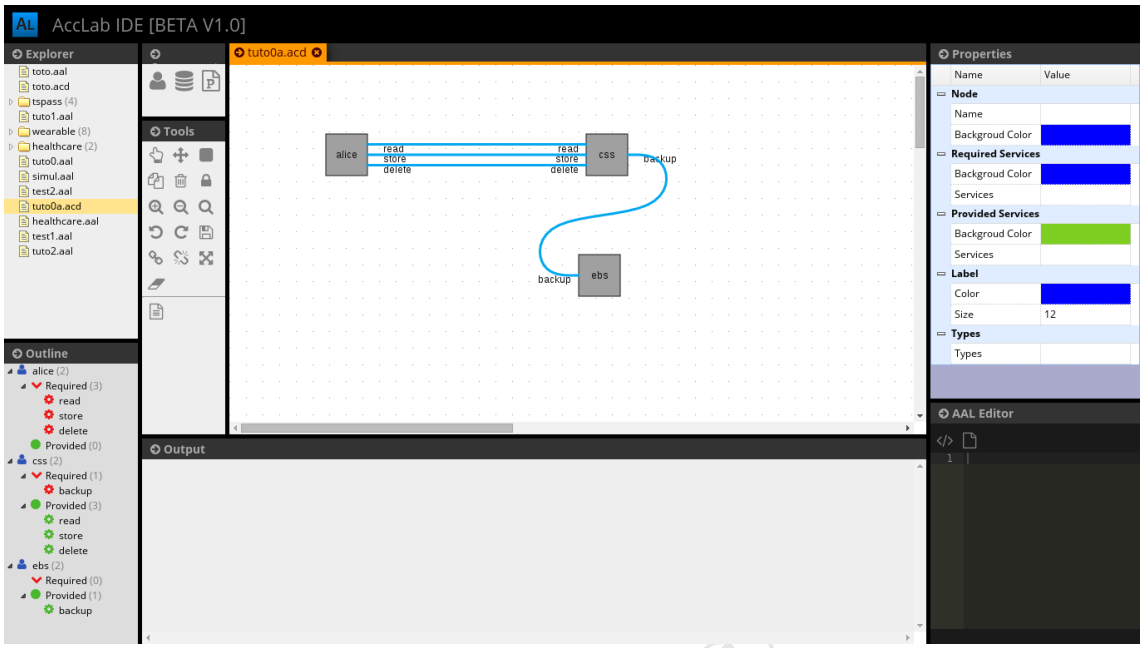

Figure 3: Editing diagram

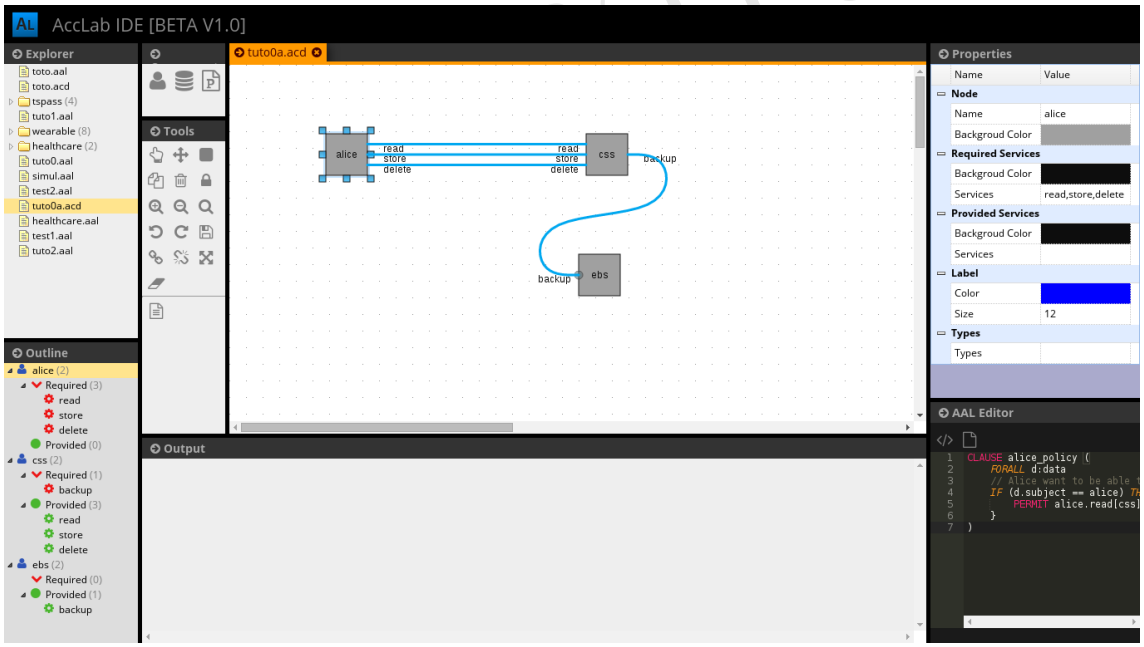

Figure 4: Adding AAL policies

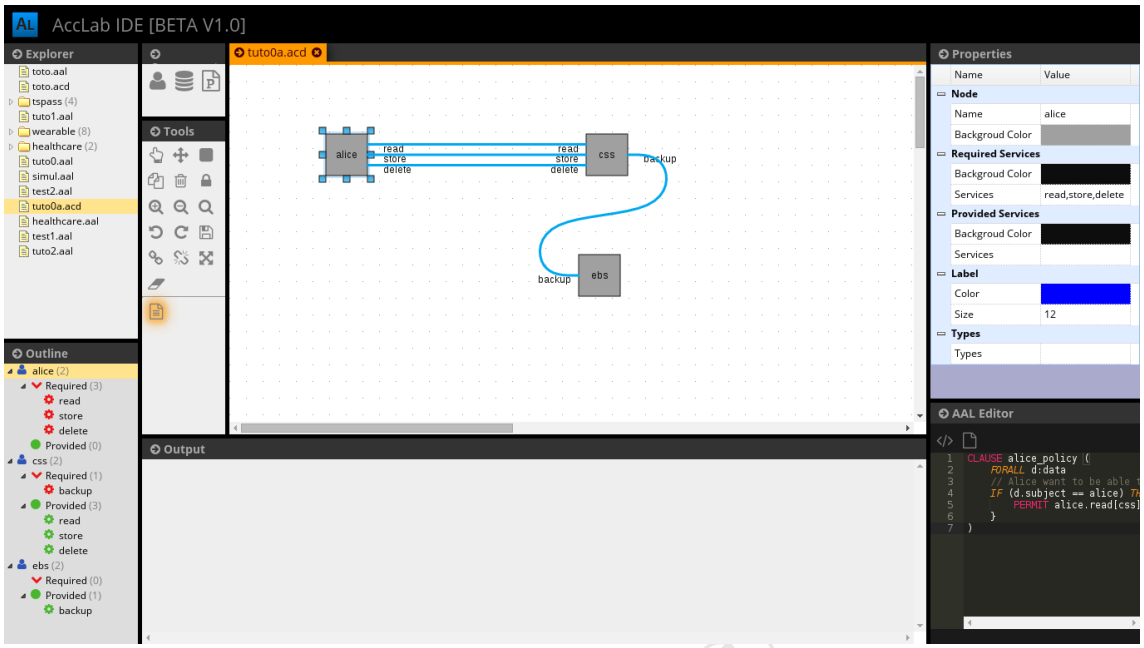

Figure 5: Generate AAL file

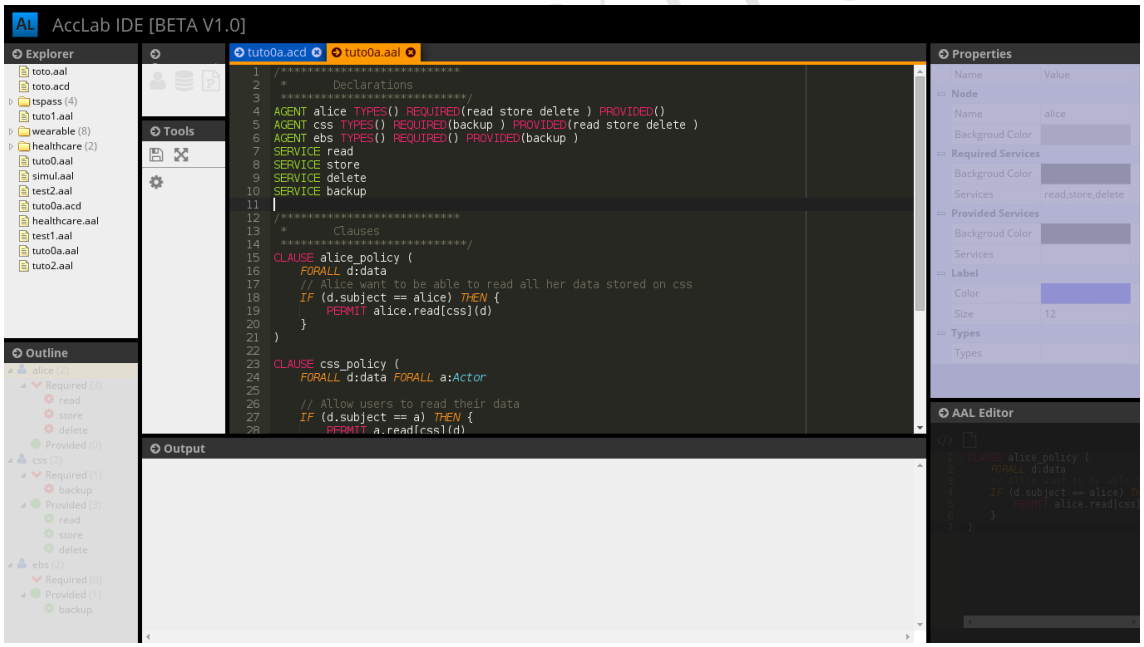

Figure 6: Generated AAL file

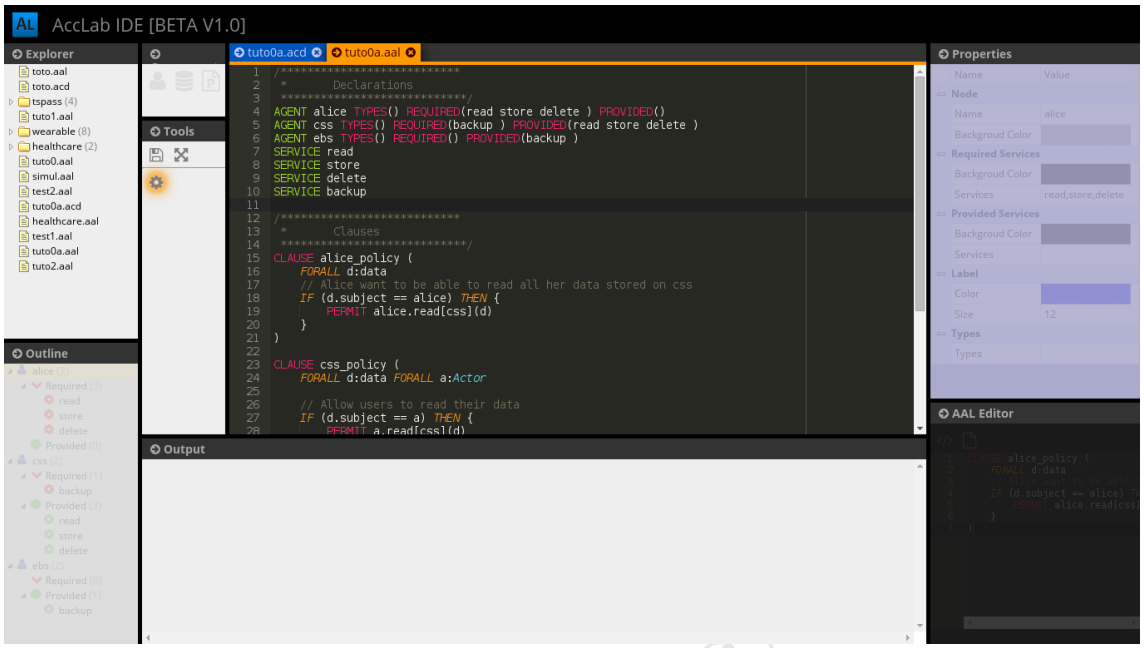

Figure 7: Compiling AAL file

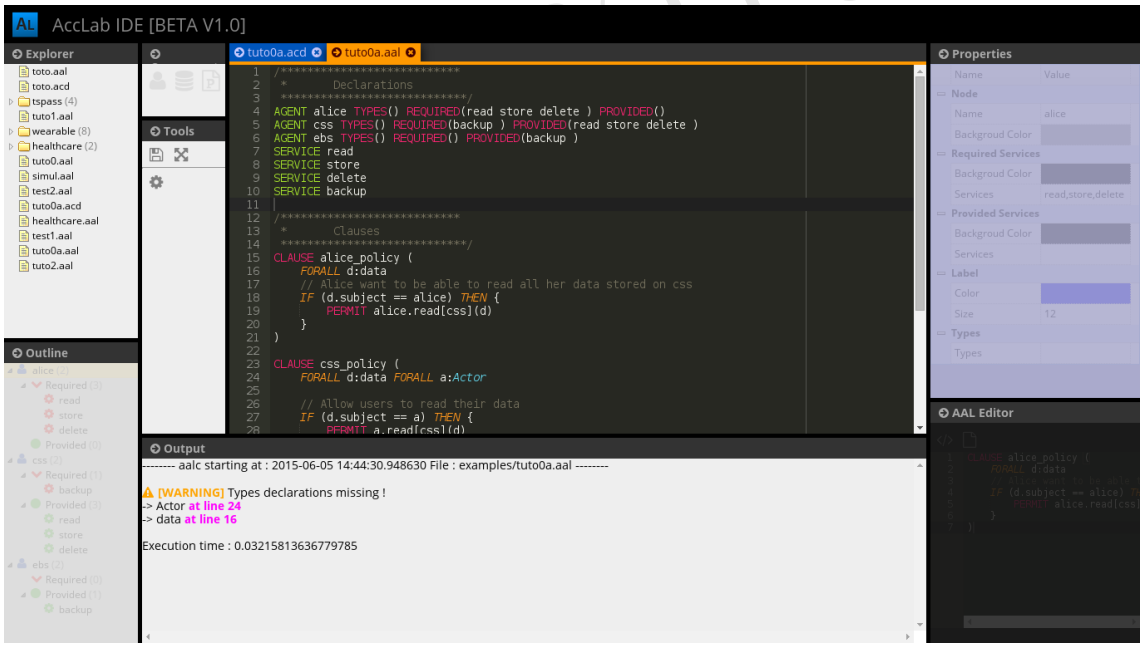

Figure 8: Result

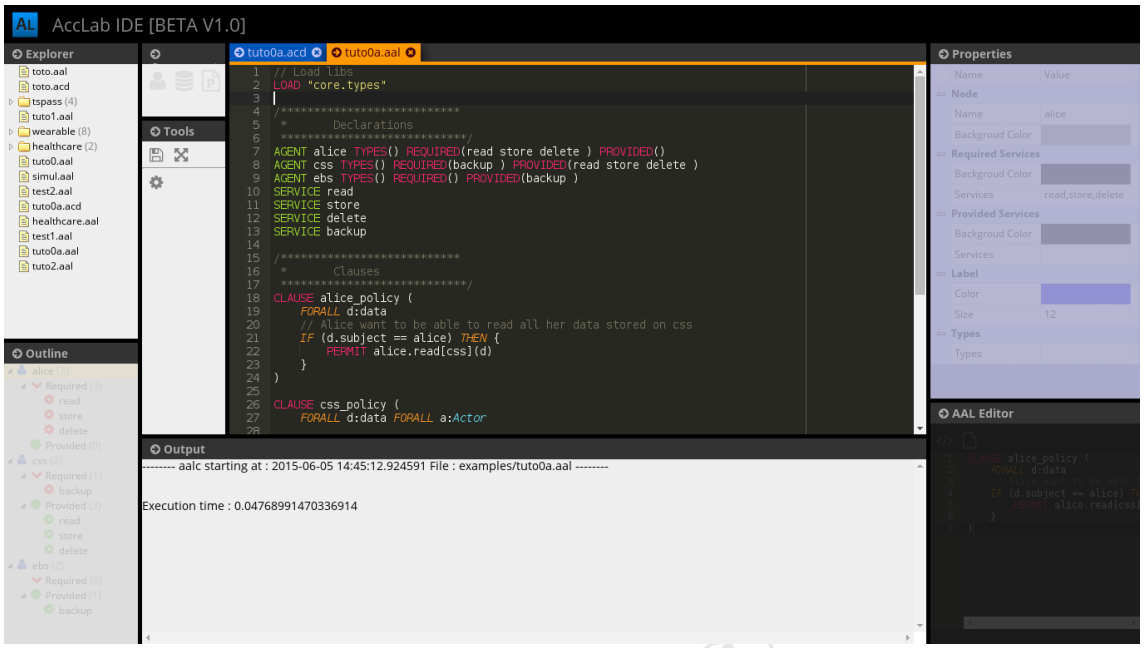

Figure 9: Load libs

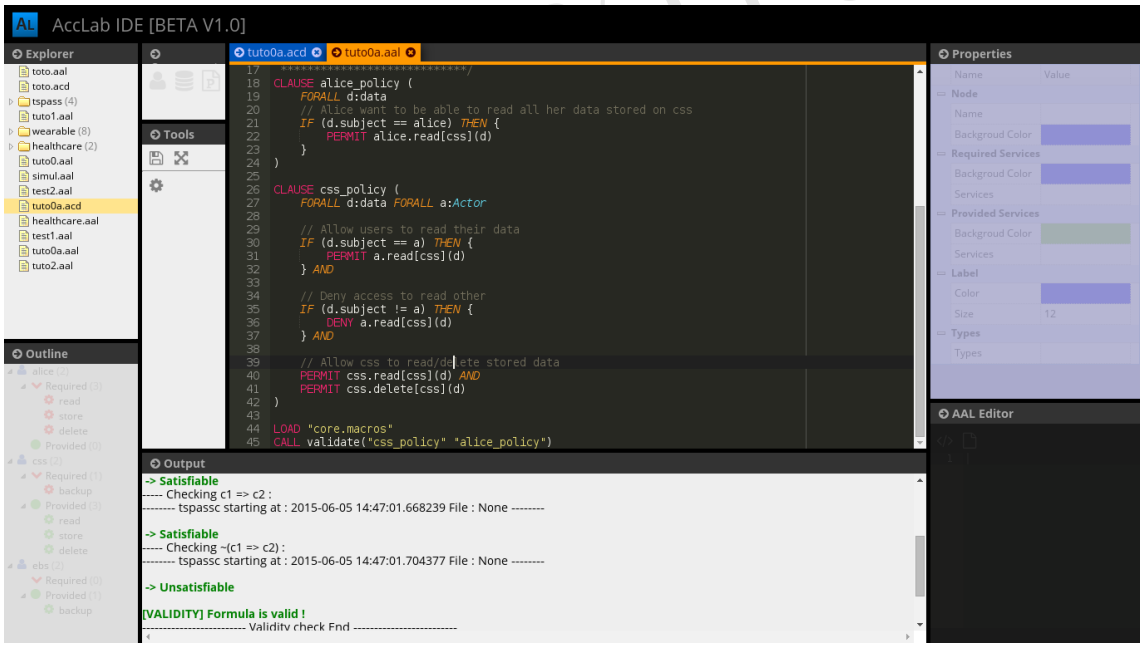

Figure 10: Validation macro