Embedding ASP in mobile systems: discussion and preliminary implementation

Francesco Calimeri Davide Fuscà Stefano Germano
Simona Perri Jessica Zangari

Department of Mathematics and Computer Science, University of Calabria, Italy

Workshop on Answer Set Programming and Other Computing Paradigms, 2015

Table of Contents

Motivation

- Ease the use of ASP [GL91, Bar03] in industrial-level and enterprise applications [CR13, LR15]
- Popularity of "smart" /wearable devices is constantly increasing
- ICT industry is moving towards the mobile scenario
- Lack of works about ASP systems natively running on mobile devices

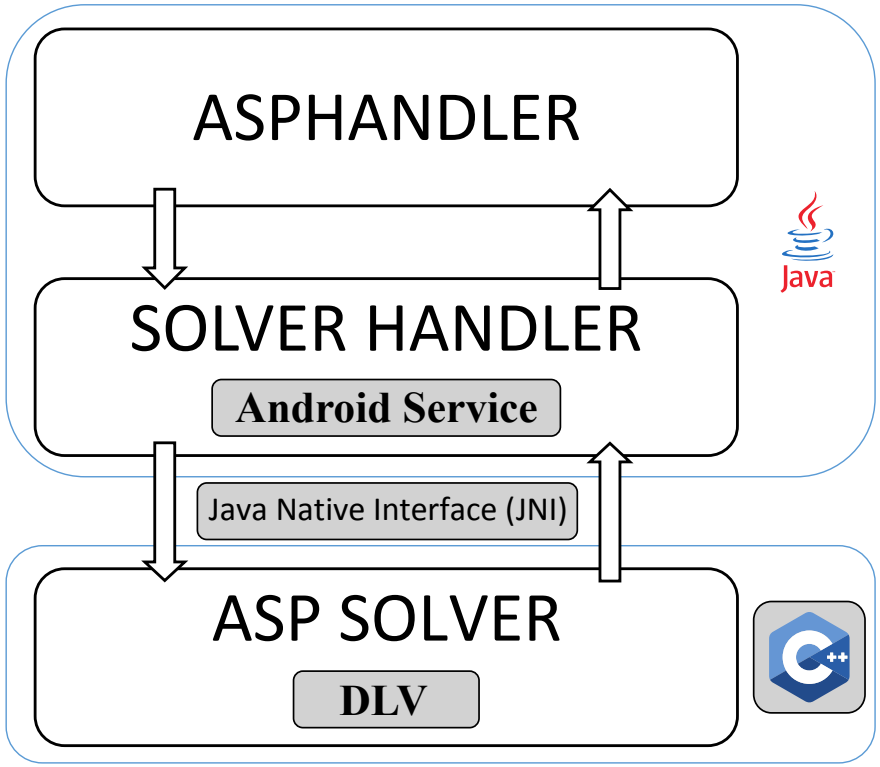
Contribution

- EMBASP: a general framework for the integration of ASP in a mobile setting
- DLVFIT: as a proof of concept, a first ASP-based Android application (actually, a health-app)

Freely available at
`https://www.mat.unical.it/calimeri/projects/embasp/`

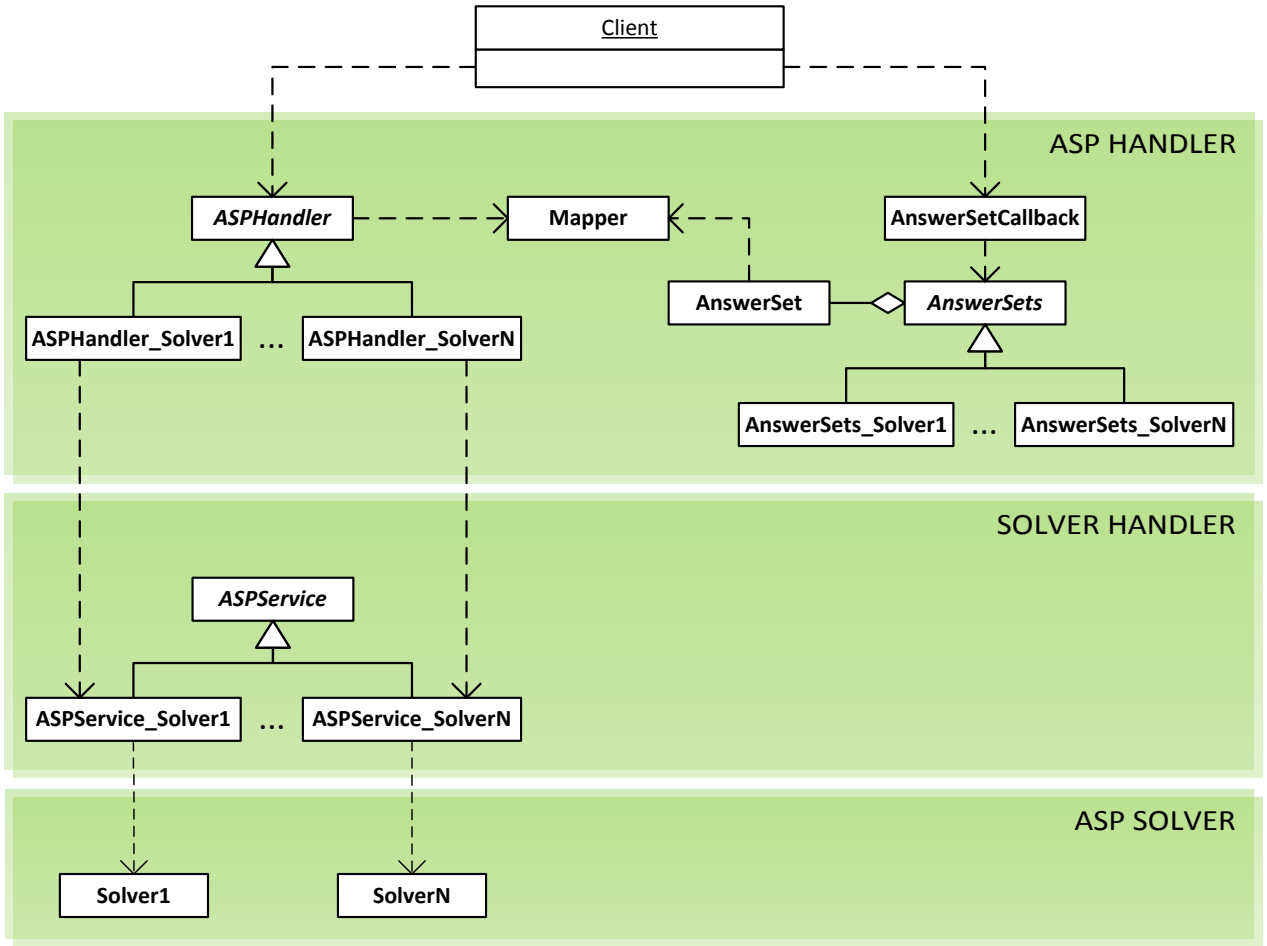
The EMBASP Framework

Abstract Architecture



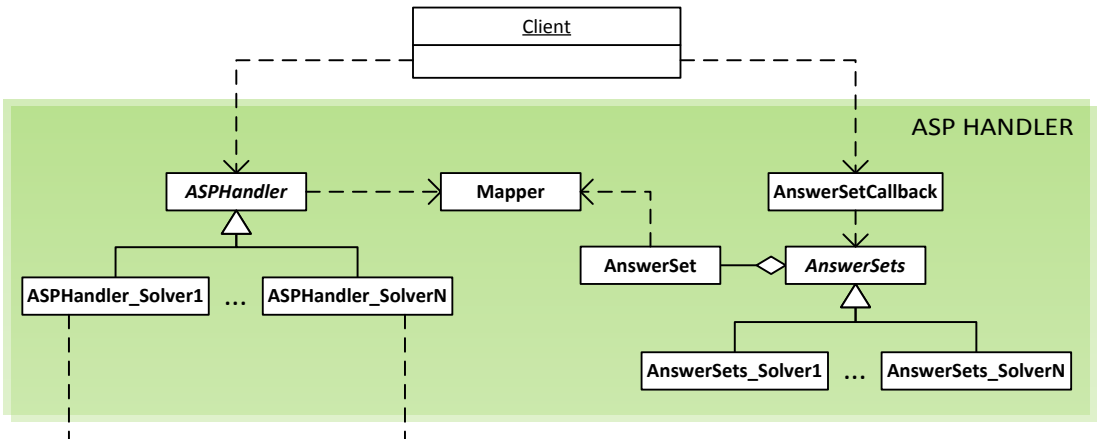
The EMBASP Framework

Framework implementation



The EMBASP Framework

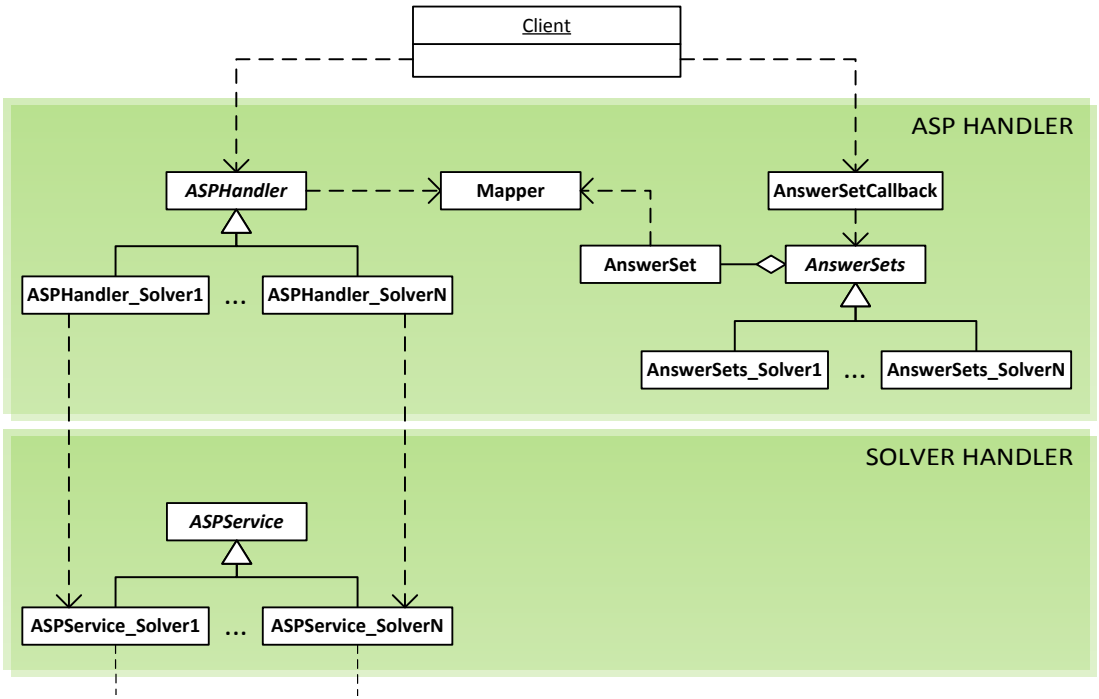
ASPHANDLER



- Provides the methods to manipulate input and output of the solvers (simple strings, files, Java Objects)
- Manages settings of all options for the actual ASP solvers
- Features proper methods for making the reasoning start

The EMBASP Framework

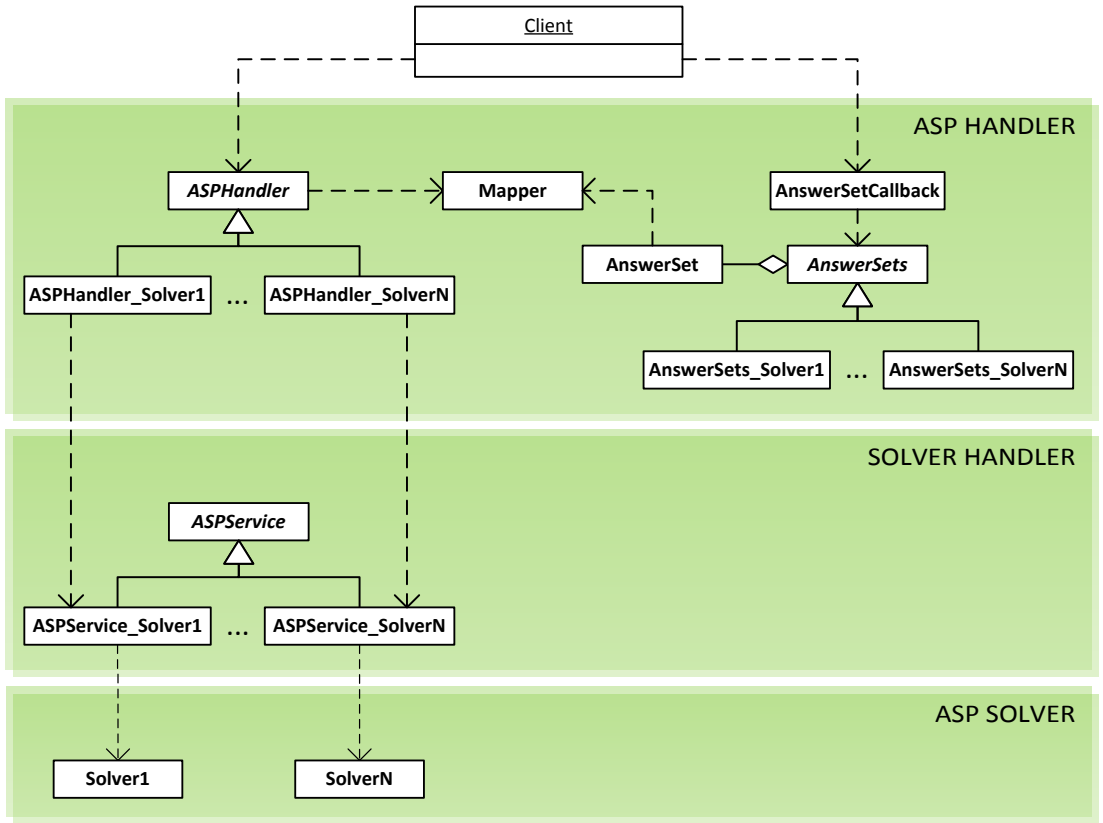
ASPHANDLER and SOLVER HANDLER interaction



- The SOLVER HANDLER layer manages invocations to the actual ASP solver(s) and gathers the results
- Asynchronous invocation (by implementing the `AnswerSetCallback`)
- Answer Sets are captured and parsed by the `AnswerSets` class

The EMBASP Framework

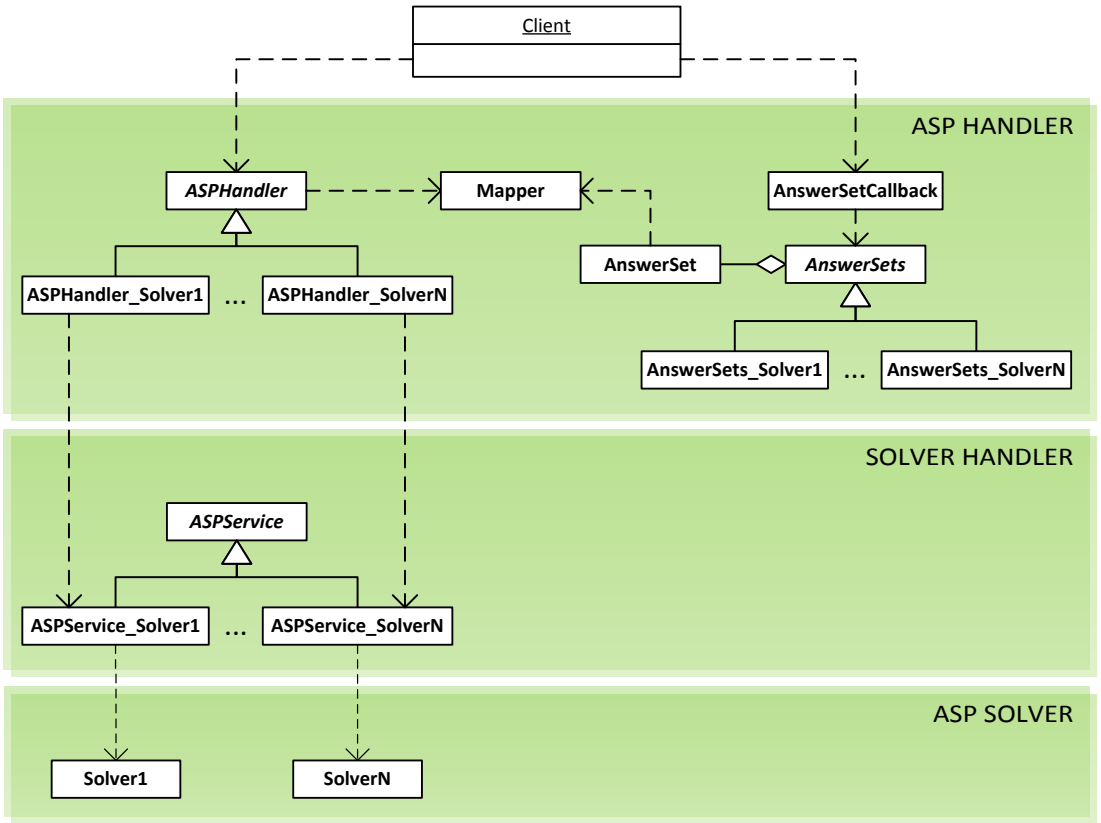
ASPHANDLER, SOLVER HANDLER and ASPSOLVER interaction



- NOT bounded to a single solver
- Different solvers can be managed directly within the framework

The EMBASP Framework

ASPHANDLER, SOLVER HANDLER and ASPSOLVER interaction



- To add an other solver:
  - Extend `ASPHandler`
  - Start a proper `ASPService`
  - Specialize `AnswerSets` to deal with the specific output of the solver

The EMBASP Framework

The Mapper

- Intended to automatically convert
  - the input for the solver from plain Java Objects into Strings
  - the output of the solvers from Strings into Java Objects
- The process is guided by proper annotations:
  - *@Predicate(string_name)*
  - *@Term(integer_position)*

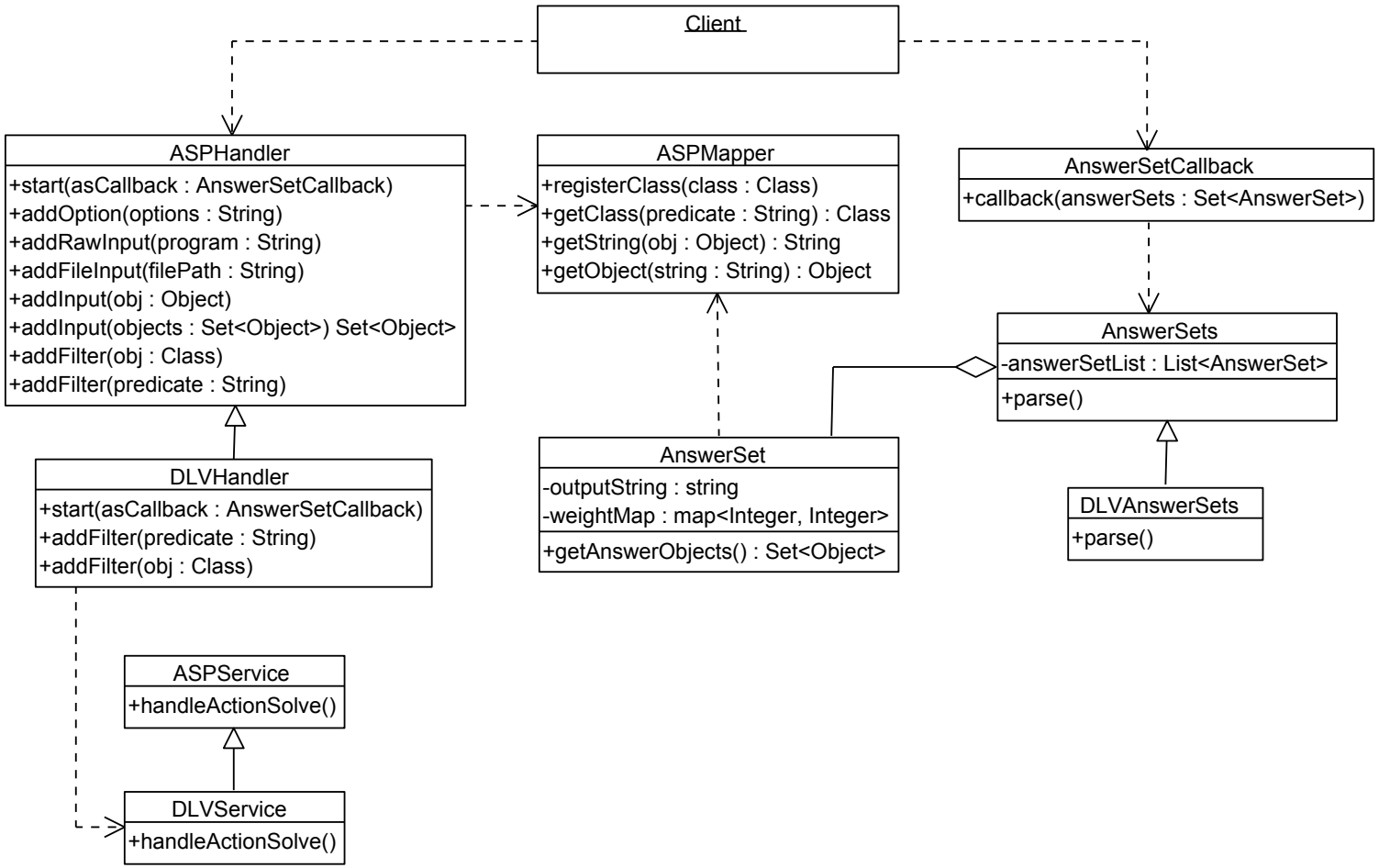
```
@Predicate("edge")
    public class Edge {
        @Term(0)
        private String node1;
        @Term(1)
        private String node2;
        ...
    }
```

```
Edge e1 = new Edge("n1","n2");
Edge e2 = new Edge("n1","n3");
```

```
edge("n1","n3").
edge("n2","n3").
```

An EMBASP Specialization

for ANDROID and DLV



The EMBASP Specialization

Technical Details

- ANDROID [Anda]
  - The most used mobile operating system worldwide, due also to its open source nature
  - The development model is currently based on the Java programming language

The EMBASP Specialization

Technical Details

- ANDROID [Anda]
  - The most used mobile operating system worldwide, due also to its open source nature
  - The development model is currently based on the Java programming language
- JNI (Java Native Interface) [JNI] and Android NDK (Native Development Kit) [Andb]
  - The use of JNI grants the access to the API provided by the Android NDK, and to the exposed DLV functionalities directly from the Java code of an Android application
  - The NDK allows developers to implement parts of an Android application as "native-code" languages, such as C and C++
  - These technologies represent the general and standard way to realize the porting of a C++ software in an Android context

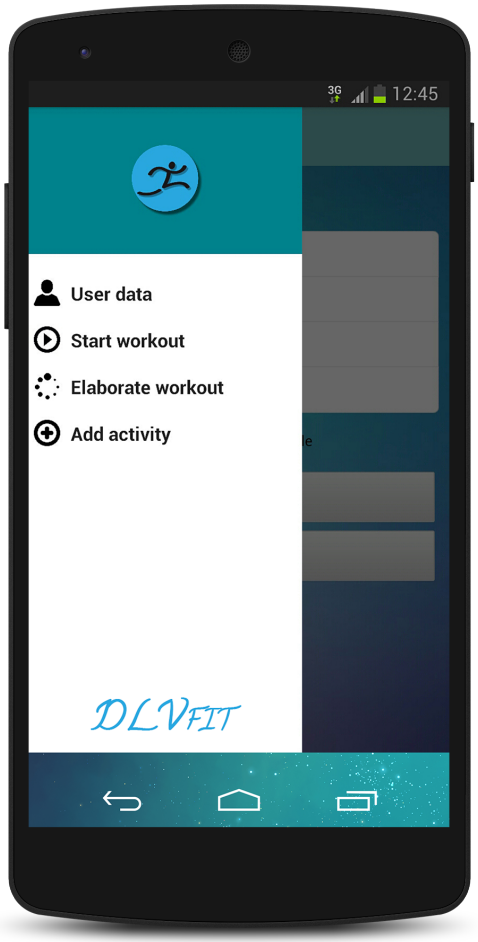
The EMBASP Specialization

Technical Details

- ANDROID [Anda]
  - The most used mobile operating system worldwide, due also to its open source nature
  - The development model is currently based on the Java programming language
- JNI (Java Native Interface) [JNI] and Android NDK (Native Development Kit) [Andb]
  - The use of JNI grants the access to the API provided by the Android NDK, and to the exposed DLV functionalities directly from the Java code of an Android application
  - The NDK allows developers to implement parts of an Android application as "native-code" languages, such as C and C++
  - These technologies represent the general and standard way to realize the porting of a C++ software in an Android context
- DLV [LPF+06]

DLVFIT

A first full native ASP-based ANDROID App



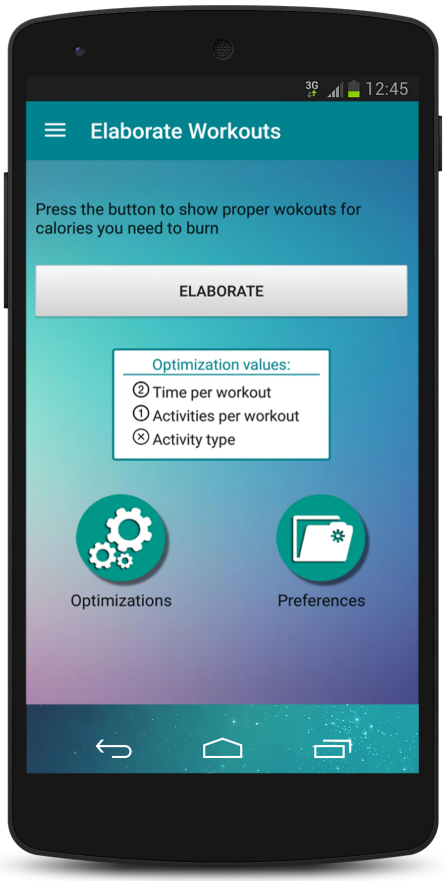
DLVFIT

Features

- The user expresses her own health goals and preferences in a customizable way, along many composable dimensions
- By means of the Google Activity Recognition APIs [Goo], in the background the app constantly detects the current user activity and stores proper information
- At any time, the user might ask for a suggestion about a workout plan for the rest of the day: this task is carried on by the reasoning module, that prepares a (set of) proper workout plans

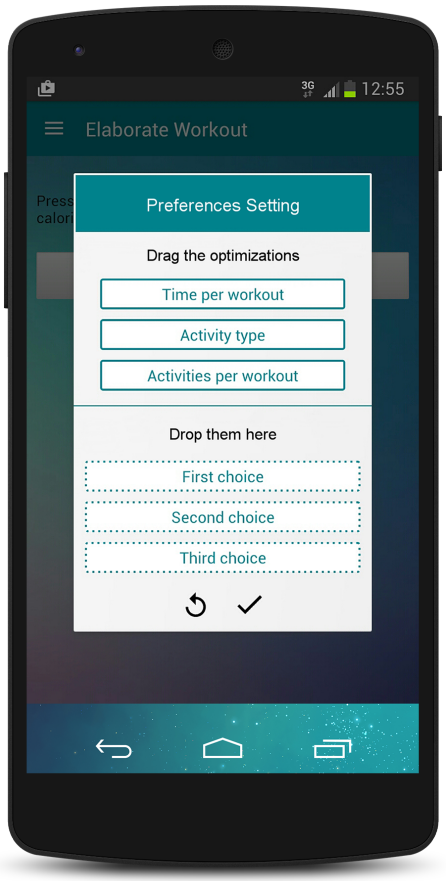
DLVFIT

Asking for a workout plan



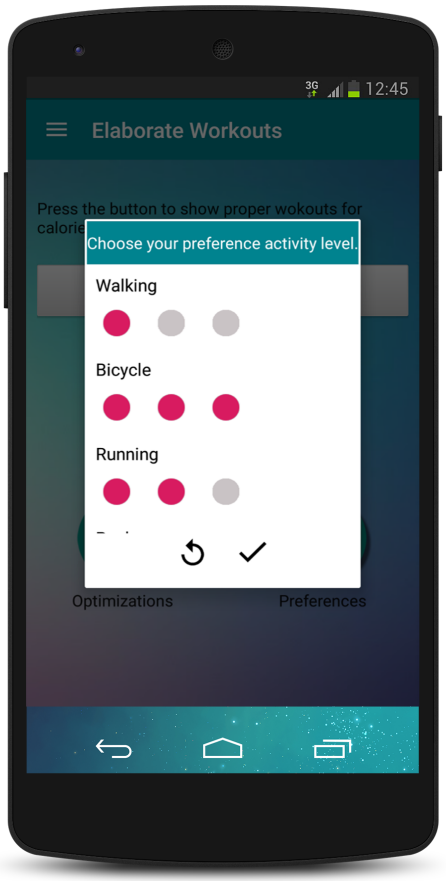
DLVFIT

Expressing priorities



DLVFIT

Expressing preferences over different workout levels



DLVFIT

ASP Reasoning Module

- The app builds dynamically a proper ASP program complying with the very personal goals and preferences previously expressed
- A classic Guess/Check/Optimize paradigm is used:
  - *Guess*: Compute how much time should be spent on each exercise
  - *Check*: Find only admissible workout plans
  - *Optimize*: Try to satisfy the user's preferences to the largest possible extent

DLVFIT

ASP program: relevants concepts

`calories_burnt_per_activity(A, C)`

the calories burnt (C), in each unit of time, per each Activity (A)

`remaining_calories_to_burn(R)`

the calories that remain to burnt in the current day

`how_long(A, D)`

the amount of the time that can be spent for each activity

`max_time(T)`

the duration of the workout

`surplus(C)`

the maximum surplus of calories to burn of the suggested workouts

`optimize(T, W, P)`

the specific optimization operation(s) that the user wants to perform

DLVFIT

An example of Input I (Basic Concepts)

```
calories_burnt_per_activity("ON_BICYCLE", 5).
calories_burnt_per_activity("WALKING", 2).
calories_burnt_per_activity("RUNNING", 11).

remaining_calories_to_burn(200).

how_long("ON_BICYCLE", 10).
how_long("ON_BICYCLE", 20).
how_long("WALKING", 10).
how_long("WALKING", 20).
how_long("RUNNING", 10).
how_long("RUNNING", 20).

max_time(20).

surplus(100).
```

DLVFIT

An example of Input II (Custom Optimizations)

```
optimize("RUNNING", 1, 3).
optimize("WALKING", 2, 3).
optimize("ON_BICYCLE", 3, 3).
```

maximize the number of favourite activities to perform

```
optimize(time,0,2).
```

minimize total time spent exercising

```
optimize(activities, 0, 1).
```

minimize total number of activities to perform

DLVFIT

Logic rules composing the ASP program

```
activity_to_do(A, HL) | not_activity_to_do(A, HL) :- how_long(A, HL).

:- activity_to_do(A, HL1), activity_to_do(A, HL2), HL1 != HL2.

:- remaining_calories_to_burn(RC), total_calories_activity_to_do(CB),
   RC > CB.

:- remaining_calories_to_burn(RC), total_calories_activity_to_do(CB),
   CB > RCsurplus, RCsurplus = RC + surplus.

:- max_time(MTS), total_time_activity_to_do(TS), MTS < TS.

:~ optimize(time, _, P), activity_to_do(_, HL). [HL:P]

:~ optimize(activities, _, P), #int(HM),
   HM = #count{A, HL : activity_to_do(A, HL)}. [HM:P]

:~ optimize(A, W, P), activity_to_do(A, _). [W:P]
```

DLVFIT

Advantages of the approach

- Wide range of customization possibilities thanks to the modelling capabilities and the declarative nature of ASP
- Flexibility and possibility to build the ASP program(s) at runtime and to customize the modules ease the developer's job of make the app comply to the user's desiderata

Related Work

Clingo4 and Java Wrapper

Clingo4 [GKKS14]

- Enables a form of control over the computational tasks of the embedded ASP solver *Clingo* with scripting languages *lua* and *python*
- The main purpose is the support of dynamic and incremental reasoning

Java Wrapper [Ric03]

- Acts like a versatile wrapper wherewith the Java developers can interact with the ASP solver (DLV)
  - Differently, EMBASP makes use of Java Annotations, allowing an easy mapping of input/output to Java Objects

Related Work

JDLV

JDLV [FGLR12]

- Based on JASP, an hybrid language that allows a bilateral interaction between ASP and Java
- Uses JPA annotations to define how Java classes map to relations, similarly to ORM frameworks
  - Differently, EMBASP exploits custom annotations, almost effortless to define, in order to deal with the mapping

Related Work

HealthyLife

HealthyLife [DLL13]

- Prototype system which makes use of ASP-based Stream Reasoning (ASR) in a mobile health app
- According to a cloud computing paradigm uses internet connections in order to communicate with the reasoning service
- The focus of *HealthyLife* is primarily to detect users daily activities and try to deal with ambiguities when recognizing situations, while DLVFIT delegates this task to Android Recognition API: its primary goal is to experiment with the usage of ASP on mobile devices

Related Work

Cloud-based Approach

- Pro
  - Grants great computational power to low-end devices
  - No need for actually porting a system to the final user's device
  - Performance is not an issue

Related Work

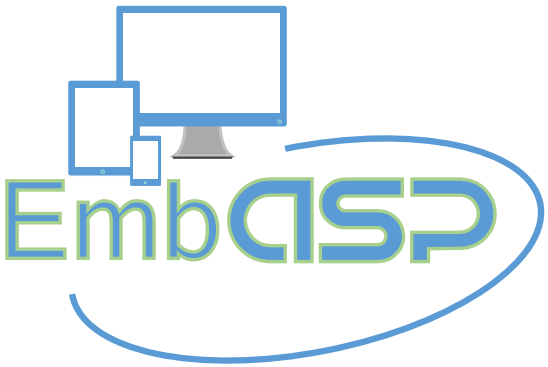
Cloud-based Approach

- Pro
  - Grants great computational power to low-end devices
  - No need for actually porting a system to the final user's device
  - Performance is not an issue
- Cons
  - Needs a proper application hosting architecture
    - set-up might be tricky if not hard
    - costs might be an issue
  - Might need a stable and fast internet connection

Conclusions

- EMBASP
  - A general framework for embedding the reasoning capabilities of ASP into external systems
  - A specialization for the mobile setting, tailored for making use of DLV within Android apps
- DLVFIT
  - First mobile app natively running an ASP system
  - Android health app that shows the effectiveness of the framework
- Future Work
  - Test the framework over different platforms and solvers
  - Evaluate the performances of our porting with detailed benchmarks over several version of Android and multiple devices
  - Further investigate the potential of ASP on mobile systems by means of new applications

Thank you for your attention.



UNIVERSITÀ DELLA CALABRIA

References I

Android.
`http://www.android.com`.

Android Native Development Kit.
`https://developer.android.com/tools/sdk/ndk`.

Chitta Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski.
Answer set programming at a glance.
Commun. ACM, 54(12):92–103, 2011.

Francesco Calimeri and Francesco Ricca.
On the application of the answer set programming system dlv in industry: a report from the field.
Book Reviews, 2013(03), 2013.

References II

Thang M Do, Seng W Loke, and Fei Liu.
Healthylife: An activity recognition system with smartphone using logic-based stream reasoning.
In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 188–199. Springer, 2013.

Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Declarative problem-solving using the dlv system.
In *Logic-based artificial intelligence*, pages 79–103. Springer, 2000.

Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner.
Answer Set Programming: A Primer.
In *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School - Tutorial Lectures*, pages 40–110, Brixen-Bressanone, Italy, August-September 2009.

Onofrio Febbraro, Giovanni Grasso, Nicola Leone, and Francesco Ricca.
JASP: a framework for integrating Answer Set Programming with Java.
In *Proc. of KR2012*. AAAI Press, 2012.

References III

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub.
Clingo= asp+ control: Preliminary report.
arXiv preprint arXiv:1405.3694, 2014.

Michael Gelfond and Vladimir Lifschitz.
Classical Negation in Logic Programs and Disjunctive Databases.
New Generation Computing, 9:365–385, 1991.

Google Activity Recognition API.
`https://developer.android.com/reference/com/google/android/gms/location/ActivityRecognition.html`.

Java Native Interface.
`http://docs.oracle.com/javase/8/docs/technotes/guides/jni`.

Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello.
The DLV System for Knowledge Representation and Reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, July 2006.

References IV

Nicola Leone and Francesco Ricca.
Answer set programming: A tour from the basics to advanced development tools and industrial applications.
In *RR2015, to appear*, LNCS, 2015.

Victor W. Marek and Mirosław Truszczyński.
Stable Models and an Alternative Logic Programming Paradigm.
In Krzysztof R. Apt, V. Wiktor Marek, Mirosław Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

Ilkka Niemelä.
Logic Programming with Stable Model Semantics as Constraint Programming Paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3–4):241–273, 1999.

Francesco Ricca.
The dlv java wrapper.
In *APPIA-GULP-PRODE*, pages 263–274. Citeseer, 2003.