ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1722

Applications of Argumentation: The *SoDA* Methodology

Nikolaos I. Spanoudakis[1] and **Antonis C. Kakas**[2] and **Pavlos Moraitis**[3]

1 Introduction

The area of argumentation is now at a stage that it can benefit from the study of systematic methodologies for building applications of argumentation. Herein, we present such a systematic argumentation software methodology, called *SoDA (Software Development for Argumentation)*, that facilitates the principled modeling of real life problems, and the *Gorgias-B* tool that supports it. *Gorgias-B* builds on the system *Gorgias* that implements the theoretical framework proposed in [1]. *SoDA* and *Gorgias-B* built on the successful use of *Gorgias* during the last ten years by different users for developing real life applications (see http://gorgiasb.tuc.gr/Apps.html).

2 Background

In [1] the authors proposed a preference-based argumentation framework where theories may be composed at different levels. The framework allows to represent arguments as rules whose heads are either alternative options (e.g. actions, decisions) or priorities over rules. An argument attacks (or is a counter argument to) another when they derive a contrary conclusion. These are conflicting arguments. A conflicting argument is admissible if it counter-attacks all the arguments that attack it. Arguments of the first level (i.e. concerning options) have to take along priority (or higher level) arguments and make themselves at least as strong as their counter-arguments.

In applications, we have **options** and **beliefs**. Beliefs refer to properties of the application problem environment. They can be decomposed, although not necessarily, into *Defeasible and Non-Defeasible* beliefs and some of the defeasible beliefs can be designated as **abducible** beliefs. Finally, we can have a **complementary or conflict** relation between the different options of the application.

In a medical access application, possible (simplified) examples of rules at the different levels are (here we are assuming that the parameter variables in the rules take values in their respective types, P is for Patient, D is for Doctor, F is for File, H is for Hospital):

$r_1(D,P,F) : denyAccess(D,P,F) \leftarrow true$
$r_2(D,P,F) : allowAccess(D,P,F) \leftarrow medical(P,F) \wedge worksFor(D,H) \wedge isIn(P,H)$
$p_1(D,P,F) : h\_p(r_1(D,P,F), r_2(D,P,F)) \leftarrow true$
$p_2(D,P,F) : h\_p(r_2(D,P,F), r_1(D,P,F)) \leftarrow treating(D,P)$
$pp_1(D,P,F) : h\_p(p_2(D,P,F), p_1(D,P,F)) \leftarrow true$

The first two rules are **object-level** rules. The next pair of rules are **priority rules** at the *first level*. The last rule is a *second level* priority rule.

If the argumentation theory can provide support both for an option literal and a complement one, then they are both credulously supported. In the case that only one option can be supported then we have a sceptical conclusion.

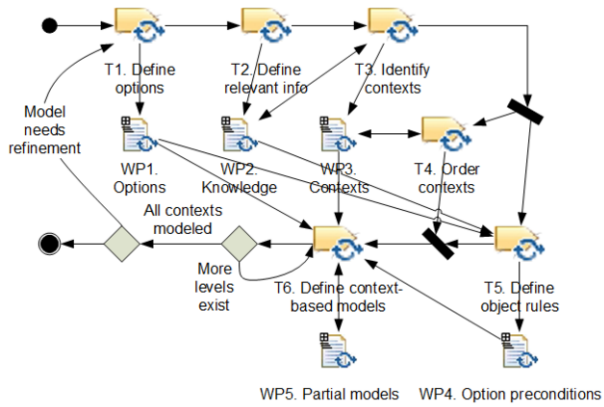

Figure 1. The *SoDA* process depicted using SPEM (Software Process Engineering Metamodel) 2.0 (http://www.omg.org/spec/SPEM/2.0/)

3 *SoDA*: Software Development for Argumentation

The *SoDA* methodology defines a high level process (presented graphically in Figure 1) requiring from the developer to consider questions about the requirements of the problem at various scenarios without the need to consider the underlying software code that will be generated. The Software Engineering Process is defined as a series of tasks (or activities) that produce Work Products (WPs). The different *SoDA* tasks (T) and their input and output WPs are:

T1: This task defines the different **options** of the application problem, including the **conflict relation** between them (written in WP1)
T2: The second task identifies the knowledge needed in order to describe the application environment (written in WP2)
T3: This task aims to separate the information in WP2 into two types: information that always exists for all instances of the problem and information that is **circumstantial**, which may be present in all instances of the problem. Circumstantial predicates are removed from WP2 and inserted in WP3. The next two tasks can be executed in parallel (T4 and T5)
T4: This task aims to sort the circumstantial information from the more general to the more specific application **contexts** in levels, starting from level one (more general contexts). Independent contexts (i.e. when the one is not a refinement of the other) can appear at the same level
T5: This task begins the process of capturing the application requirements. It aims to define for each option, O_i, the different problem environments, i.e. the sets of **preconditions**, C_i, in terms

[1] Technical University of Crete, Greece, email: nikos@science.tuc.gr
[2] University of Cyprus, Cyprus, email: antonis@cs.ucy.ac.cy
[3] LIPADE, Paris Descartes University, France, email: pavlos@mi.parisdescartes.fr

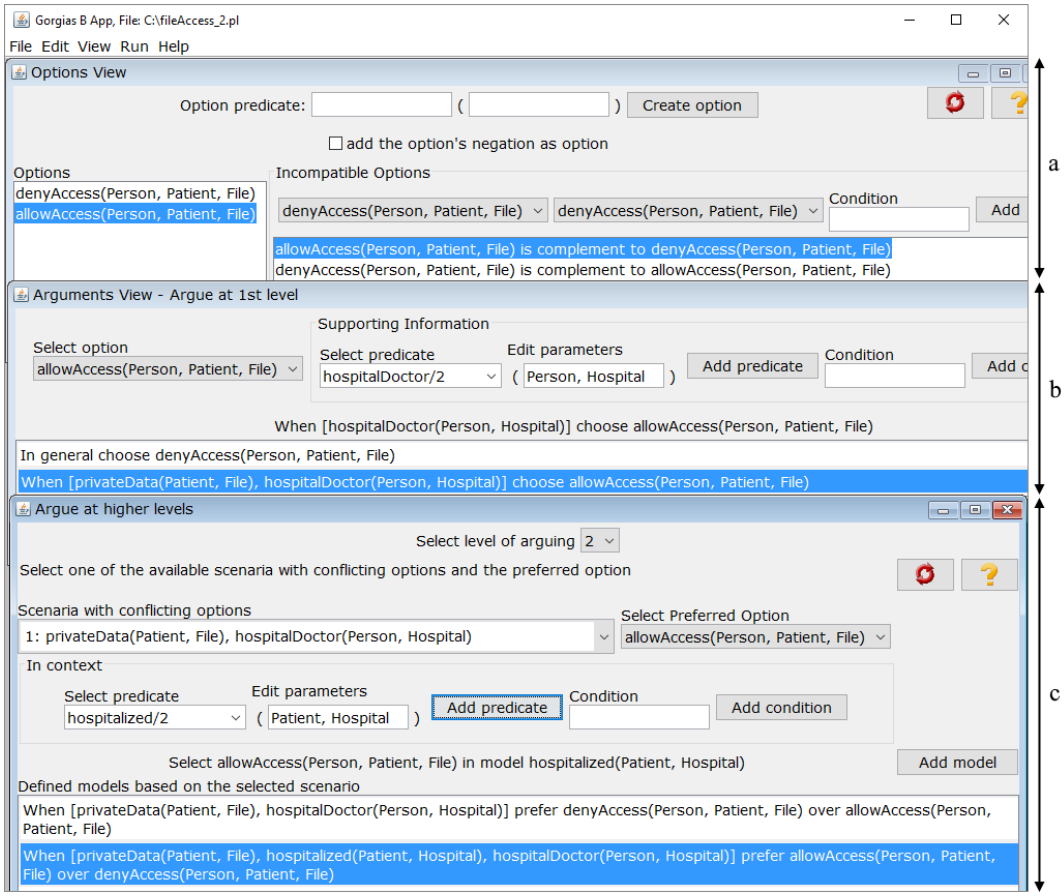

Figure 2. A screenshot of the *Gorgias-B* tool with the options view (a), the arguments view (b) and the argue view (c)

of non-circumstantial predicates appearing in WP2, where the **option is possible**. Its output, WP4, contains all such sets of preconditions

T6: This final task iteratively defines sequences of increasingly more specific **partial models or scenarios** of the world (stored in WP5) and considers how options might win over others. This starts with information from WP4 to precondition the world and iterates getting each time contextual information from the next level in WP3. At each **level** of iteration it defines which option is stronger over another under the more specific contextual information. In the final iteration, the winning options (if they exist) for each partial model are defined without extra information

4 Tool Support for *SoDA* Methodology

We have developed a new tool, *Gorgias-B* [4], to support the development of applications of argumentation under *Gorgias*, following the *SoDA* methodology, in a way that allows users with little or no knowledge of argumentation to model their application domains. Using *Gorgias-B* the user can automatically generate source code in the form of an application argumentation theory in the *Gorgias* framework. Moreover, this theory can be executed under *Gorgias-B* by specifying scenarios of interest and asking which options are credulous or sceptical solutions. The tool returns these together with the admissible arguments that support them. *Gorgias-B* also allows to specify some predicates as *abducible*, and the tool can find scenario conditions under which an option will be a solution.

In the *Options View* (Figure 2(a)), the user defines the different options. Then, the user can edit preconditions (WP4) for options in the *Argument View* (Figure 2(b)). Here the user is building the (object-level) arguments for the various options. The *Argue View* (Figure 2(c)) appears as soon as the user clicks the "Resolve conflicts" button. Here the user selects among scenarios with conflicting options more specialized cases (if they exist) where one of the other option can win. When this happens the contextual information of both previous level scenarios is combined to a new more specific scenario and the user can then repeat to select contexts in the new current level, which is always visible at the top of the dialog window.

[4] http://www.amcl.tuc.gr/gorgiasb

ACKNOWLEDGEMENTS

We acknowledge the collaboration with the Resilient Information Systems Security (RISS) group at Imperial College in the study of the application of SoDA to cyber security and data access and data sharing problems.

REFERENCES

[1] Antonis C. Kakas and Pavlos Moraitis, 'Argumentation based decision making for autonomous agents', in *Proc. Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, Melbourne, Australia*, pp. 883–890, (2003).