ADT2AMAS: Managing Agents in Attack-Defence Scenarios

Jaime Arias
LIPN, CNRS UMR 7030,
Université Sorbonne
Paris Nord
Villetaneuse, France
arias@lipn.univ-paris13.fr

Wojciech Penczek
Institute of Computer
Science, Polish Academy
of Sciences
Warsaw, Poland
penczek@ipipan.waw.pl

Laure Petrucci
LIPN, CNRS UMR 7030,
Université Sorbonne
Paris Nord
Villetaneuse, France
petrucci@lipn.univ-paris13.fr

Teofil Sidoruk
1Institute of Computer
Science, PAS
2Warsaw University
of Technology
t.sidoruk@ipipan.waw.pl

ABSTRACT
Expressing attack-defence trees (ADTrees) in a multi-agent setting allows for studying a new aspect of security scenarios, namely how the number of agents and their task assignment impact the performance of attacking and defending strategies executed by agent coalitions. Our tool ADT2AMAS allows for transforming ADTrees into extended asynchronous multi-agent systems and computing an optimal schedule with the minimal number of agents. ADT2AMAS is integrated within the graphical verification platform CosyVerif, but can also be run standalone.

KEYWORDS
asynchronous multi-agent systems; scheduling; attack-defence trees

ACM Reference Format:

1 INTRODUCTION AND MOTIVATIONS
Security of safety-critical multi-agent systems [24] is one of the major challenges. Attack-defence trees (ADTrees) [8, 15] have been developed to evaluate the safety of systems and to study interactions between attacker and defender parties. They provide a simple graphical formalism, where nodes represent possible attacks against a system and defences employed to protect it. However, the defenders might forgo a costly defence, or a defence could take too much time and thus fail. The attack to be achieved is the root one, which requires all (for the AND nodes) or part (for the OR nodes) of those in its subtrees to be successful.

Recently, it has been proposed to model ADTrees in the formalism of asynchronous multi-agent systems (AMAS) extended with certain ADTree characteristics [4, 21]. In this setting, one can reason about attack/defence scenarios considering agent distributions over the tree nodes and their impact on the feasibility and performance (quantified by metrics such as time and cost) of attacking and defending strategies executed by specific agent coalitions. Both the number of agents available and their distribution over the ADTree nodes affect these performance metrics. Hence, there arises the problem of an optimal scheduling, that is, obtaining an assignment that guarantees the lowest possible execution time while using the minimum number of agents required for an attack to be feasible. The tool ADT2AMAS implements the algorithm of [7] that finds such an assignment for a given ADTree. As the ADTree formalism introduces unique caveats, our approach differs from that of classical process scheduling [18, 19, 22]; to the best of our knowledge, this is the first work dealing with agents in this context. Notably, the scheduling algorithm optimises both the number of agents and the attack time, and runs in quadratic time wrt. the number of nodes. However, it is applied to a number of models that is exponential in the number of OR and defence nodes.

2 FORMAL BACKGROUND
A well-known formalism [10, 11, 14, 16, 17, 20, 23], ADTrees model security scenarios as an interplay between two opposing parties. Figure 1 depicts basic constructs, [4] gives a comprehensive overview. Attacking and defending actions are depicted in red and green, respectively. The leaves represent individual actions at the highest level of granularity. Different types of gates (available in both attack or defence types) allow for modeling increasingly broad intermediary goals, all the way up to the root, which corresponds to the overall objective. OR and AND gates are defined analogously to their logical counterparts. SAND is a sequential variant of the latter, i.e. the entire subtree $a_i$ needs to be completed before handling $a_{i+1}$. Countering actions can be expressed using gates CAND (counter defence; successful iff $a$ succeeds and $d$ fails), NODEF (no defence; successful iff either $a$ succeeds or $d$ fails), and SCAND (failed reactive defence; sequential variant of CAND, where $a$ occurs first). ADTree nodes may have numerical attributes, e.g. the time needed to perform an attack, or its financial cost. Conditions are boolean functions over the attributes, used as constraints to counter-defence nodes.
The translation of ADT2AMAS to extended asynchronous multi-agent systems (EAMAS, [4, 12, 13]), called also models, was proposed in [4, 21]. To that end, the semantics of an asynchronous MAS of [12, 13] was extended to account for the node attributes and conditional constraints. Each ADTree node corresponds to a separate automaton, with specific patterns used for different types of nodes. Transitions are decorated with actions and the following elements: (i) a message \( f_{\text{msg}}(t) \in \{!?, ?\} \times M \cup \{\lambda\} \) indicating whether there is a synchronisation on a message or the action is local; (ii) a guard \( f_{\text{guard}}(t) \in \text{Guards} \) constraining the transition; (iii) an update function \( f_t : AT \rightarrow \text{EXP}(AT, \text{FUN}) \) expressing how the transition modifies the attributes. In this setting, groups of agents working for the attacking and defending parties can be considered. Note that the feasibility of an attack is not affected by the number or distribution of agents over ADTree nodes, while some performance metrics, such as time, clearly depend on both factors (e.g., a lone agent can sequentially handle all the actions, albeit usually much slower). Furthermore, the success or failure of particular defence nodes results in a number of unique configurations, leading to multiple model variants to be analysed. Consequently, the optimal distribution of agent coalitions is of vital importance for both parties, allowing them to prepare for multiple scenarios, depending on how many agents they can afford to recruit. Thus, synthesising an assignment that achieves a minimal execution time using the least possible number of agents is a problem of high interest and importance.

3 ARCHITECTURE AND TECHNOLOGY

ADT2AMAS [5] is an open source tool written in C++17. It allows for: (1) translating an ADTree into an EAMAS, and (2) computing a minimal scheduling with a minimal number of agents to make the root node’s task successful. The ADTree can be specified using the provided API, or loaded from a simple-syntax text file (Figure 2a).

For (1), the tool generates a \( \LaTeX \) file of the EAMAS, and an .imi file containing its specification, which can be read by the model-checker IMITATOR [3] in order to run automatic verifications. For (2), ADT2AMAS generates a \( \LaTeX \) file of each tree transformation needed to compute the minimal assignment, and an ASCII table of the minimal scheduling. Compiling \( \LaTeX \) to PDF provides a visual feedback of the EAMAS and of the scheduling algorithm’s steps.

ADT2AMAS is also integrated within the CosyVerif [2] formal specification and verification environment [2], thus providing a multi-platform, user-friendly, zero-configuration tool (Figure 2b). Its JavaScript web interface, CosyDraw, allows for designing formal models extensible via FML (Formalism Markup Language) and GrML (Graph Markup Language) specifications. Furthermore, it supports any tool providing a SOAP web service. In our case, we extended the graphical interface with the ADTree elements (Figure 2c) and implemented a web service offering the algorithms supported by ADT2AMAS [6]. The tool can be accessed at https://cosyverif.lipn.univ-paris13.fr, and its video demonstration at https://youtu.be/DGLtU5P-ao8.

4 EXPERIMENTS

Selected experimental results are provided in Table 1, with additional ones available on the tool web page [1]. Subsequent columns denote: (1) the case study, (2) the number of possible defence configurations, which influence the number of models to consider, (3) the number of a model variant; note that some combinations of defences result in the same model, or do not allow for a successful attack, (4) the minimal number of agents obtained by our scheduling algorithm, (5) the minimal attack time. The tool also provides a corresponding schedule of agents’ actions (not shown here), and generates the resulting EAMAS (currently, for 1 agent only), which can then be passed to the IMITATOR model checker. This allows for the verification of other desired properties (e.g., a model with a higher attack time might be preferable due to its far lower cost).

Table 1: Experimental results on case studies

<table>
<thead>
<tr>
<th>case study</th>
<th># def</th>
<th>no. of model</th>
<th># agents</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>treasure</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>125 min</td>
</tr>
<tr>
<td>forestall</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>43 days</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>54 days</td>
</tr>
<tr>
<td>iot-dev</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>53 days</td>
</tr>
<tr>
<td>gain-admin</td>
<td>16</td>
<td>1</td>
<td>2</td>
<td>2942 min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>4320 min</td>
</tr>
<tr>
<td>scaling</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5762 min</td>
</tr>
</tbody>
</table>

5 FUTURE WORK

Currently, model specifications (.imi files) including more than one agent are prepared by hand. We plan to improve the integration of ADT2AMAS with IMITATOR by a fully automatic generation of EAMAS for an arbitrary number of agents. Furthermore, ADT2AMAS can be integrated with other model checkers such as UPPAAL [9].
REFERENCES