Goal Modelling Meets Service Choreography: A Graph Transformation Approach

Michalis Bachras
Dept. of Eclectrical & Comp. Engineering
National Technical Univ. of Athens
Athens, Greece

Kostas Kontogiannis
Dept. of Computer Science
Western University
London ON. Canada

***Abstract*—As microservices become one of the predominant architectural styles for distributed enterprise computing, there is a need to devise frameworks which allow for the goal driven composition and coordination of such highly granular service components. Even though a number of service composition and orchestration techniques have been proposed over the past decade, these do not take into account stakeholders' intents as well as data, control, and temporal interdependencies between actions microservices can perform. In this paper, we present extensions to goal models with respect to data, logical and temporal dependencies exhibited between tasks and actions among microservices, and we propose a framework based on a graph transformation approach which, when applied to the extended goal models, can yield service invocation plans that achieve the desired requirements and constraints denoted by the specific goal models being considered.**

***Keywords*-Goal models, service composition, service choreography, microservices.**

I. Introduction

With the proliferation of infrastructures, such as mobile and cloud infrastructures that support dynamic provision of services and resources, there is a dire need to bridge the gap between the intended system requirements, the functionality of these available services, and the stakeholders' goals. The problem is amplified due to the fact that in such systems specific service functionality can now be offered by different highly granular containerized microservices, and the fact that usually many stakeholders with diverse and often conflicting requirements, are involved. Examples of different stakeholders involved are the system's end-users, certification bodies, regulatory authorities as well as logging and monitoring systems which have to respect key privacy and security requirements. The software engineering community has responded by proposing a number of requirements specification and conceptual modeling frameworks for enterprise software systems and services. These frameworks provide valuable tools for denoting user's intentions and system requirements. One such framework is based of the i^* and the Goal Modelling formalisms [1]. In this paper, first we propose extensions to the goal model metamodel for capturing structural and temporal dependencies between user intents, service goals, data resources, tasks, and actions required to achieve the system's functional and non-functional requirements, and second we present a collection of transformations whereby the extended Goal Models are transformed to a collection of graphs referred to a Action Dependency Graphs or ADGs [2] [3]. The ADGs are topologically ordered to yield alternative actions plans. These compiled action plans can serve as templates which can be instantiated by specific microservices.

As a motivational example, let us consider an e-commerce company which utilises a microservices architectural model. In such a model, highly granular components (i.e. microservices) implement different functionalities related to how and what order the goods are presented to the customer, how discounts on different products are applied, how payments are processed and reconciled on the back-end, how shipping agreements with various operators (courier services, postal service, freelance contractors) are processed, and how customs brokerage services are selected. In this context, there may be many different shipping operator services implemented as different microservices, many different customs brokerage services, and many different payment clearing services available in such an ecosystem. Each such service provides unique properties and fulfills different requirements (cost, security, processing time guarantees etc.). Furthermore, these services have to be orchestrated, so that they conform with temporal, logical, and resource dependencies and constraints. For example, certain goods may have to clear customs within five hours while other goods may be prohibited on some transportation means (e.g. air-transport). The problem then is given the various available services (courier, brokerage, customs, payment clearance) and which may possibly have conflicting requirements (e.g. cost vs. speed of processing), compose a workflow as a sequence of actions, which will satisfy all requirements along with any other temporal, logical, or resource constraint and at the same time satisfy the top goal.

This paper is organised as follows. Section II presents related work. Section III provides and an outline of Goal Models and presents the proposed Goal Model extensions. Section IV presents the transformations from the extended Goal Models to ADGs supplemented a running example, while Section V presents performance statistics by applying the algorithm to sizeable Goal Models. Finally, Section VI concludes the paper and discusses future research.

II. Related Work

We could say that the related proposed approaches fall into three major categories. The first category relates business modeling and enterprise application planning. The second category relates to service discovery and selection using requirements and QoS criteria, while the third category relates to service composition and orchestration.

Business and Enterprise Modeling. In [4] the authors use a rule language based on the Event-Condition-Action (ECA) paradigm in order to alter a business process and produce variant processes which fulfill specific business requirements. The main differences from our work is that we utilise an extended Goal Model instead of ECA rules, and that we produce action plans which take into account data dependencies, logical dependencies, temporal constraints as well as positive and negative contribution links between possible system operations (i.e. actions). Markovic and Kowalkiewicz in [5] present an ontology language for denoting business goals which are consequently linked with services which can be used to implement such goals. The proposed language is founded on the entities of goal, subgoal, description, measure, deadline, priority, and level of goal satisfaction. The proposed language is a subset of the Goal Model meta-model, but could also serve as a language which our approach could use to generate action plans modulo temporal constraints and parallel compositions. In [6] the authors present a methodology to associate Business Processes denoted in BPMN with stakeholder objectives denoted as Goal Models. The approach allows for the dynamic analysis and evaluation of such relationships in a quest to verify that business processes conform with policies and user requirements. The authors utilise KAOS as a their Goal modelling framework. The main differences with our work is that first we extend the Goal Model meta-model and second we use the extended Goal Model to generate consistent with it action plans (e.g. business or service workflows), as opposed to verify the compliance of flows with the Goal Model.

Service Discovery and Selection. In [7] the authors propose an approach which is based on Genetic Algorithms and a technique referred to as quality constraints decomposition in order to decompose the constraints of a complex service into simpler constraints of the constituent services. Once such a decomposition is achieved, then a search algorithm identifies the services which can optimally satisfy these simpler constraints. The approach is focusing on satisfying global constraints by optimizing local ones. In [8] Wang et al. discuss an adaptive service composition technique which aims to tackle the problem of changing QoS properties in dynamically altered environments such as mobile computing environments. The approach is based on the use of reinforcement learning and Skyline computing as a search optimization technique. The result is a service selection and composition environment which can adapt its operation in environments where QoS properties for different services are constantly changing. Zo et al. [9] propose the use of a multi-criteria Genetic Algorithm to identify a Pareto-optimal collection of services when various QoS requirements are considered. The technique is using a combination of weights and Genetic Algorithms in order to explore the QoS criteria space and reach an optimal combination of services which meet the specified service selection criteria. In [10] the authors present an approach that utilises semantic web languages such as OWL in order to annotate services and consequently select in an intelligent manner such services from service registries. The authors distinguish between three types of registries, namely ontology registries, QoS registries and general service registries. Service compositions are specified using abstract models which are instantiated by selecting appropriate services and generating executable workflow specifications.

Service Composition and Orchestration. In [11] the authors present a goal driven framework based on the *Map* metamodel to capture requirements of complex service compositions when different organizations embark into collaborations integrating in an ad-hoc manner various service components. The metamodel allows for eliciting requirements and denoting service distribution and orchestration properties. The fundamental premise in this approach is that intentions can be achieved by strategies, forming thus a directed graph which can be instantiated by concrete services in order to compose a more complex service. In [12] the authors present a goal driven service composition approach capable of exhibiting adaptive behavior through a backtracking and feedback mechanism. The approach is based on the compilation of a Goal Abstract Graph which can be instantiated to yield concrete compositions using a service discovery algorithm that takes into account QoS properties and context. The main differences of the Goal Abstract Graph with our approach is first that we focus on deducing action plans (i.e. microservice workflows) as opposed of instantiating the existing plans, and second we consider additional operators such as parallel composition, time precedence and timeouts. Orriens, et al. [13] present a rule based approach for generating service business flows that meet business objectives and user requirements. The authors propose a metamodel for service flow specification that is founded on the concepts of provider, role, activity, flow, condition, event and message, while they differentiate on rule types such as structure rules, data rules, exception rules, and constraint rules in order to capture various facets of service flow composition. The main differences with our work is that we utilise extended goal models, as opposed rules, and action dependency graphs as the underlying formalism for workflow construction. In [14] Jiang et al. present a model-driven approach to service composition whereby user requirements, user intentions, and tasks are specified using the i^* language. The specification is then transformed into an abstract Web service composition specification which through instantiation yields the final concrete service. Our approach is also using a formalism akin to i^* but allows for more complex dependencies to be taken into account on the formulation of action plans. These dependencies include positive and negative contributions, resource dependencies, temporal dependencies, and parallelism.

III. Goal Models and Extensions

Goal models refer to a conceptual modelling formalism proposed for representing requirements and intentions of agents, actors, and systems. Since their introduction [15] [16] [17], they have been widely used in the field of Software Requirements Engineering due to their simplicity and rich semantics. Goal models formulate the goals of a software system in a tree-structure form. Each goal node can be analyzed through AND/OR decomposition into subgoals. A goal is AND-decomposed if in order to satisfy it, the children-goals must be all satisfied. On the other hand, in the case of a OR-decomposed goal, at least one of the children-goals must be satisfied in order to satisfy the parent goal. For the action composition problem, we consider two more types of nodes *Task* nodes and *Action* nodes as proposed in [2] [3]. A *Task* is considered as an abstraction of a complex operation which can be AND/OR decomposed into simpler *Tasks* and finally, into *Actions*. *Actions* are the simplest nodes and they represent atomic operations that are necessary to be performed in order to fulfill complex *Tasks*.

Action nodes may require data *Resources* in order to execute. *Action* nodes can only be AND-decomposed into sets of required *Resource* nodes. In this respect, a *Resource* node represents an input parameter required by an *Action*. A *Resource Dependency link (rd)* between an *Action* and a *Resource* denotes that the *Action* node (i.e. source node of the *rd* link) generates the specific *Resource* (i.e. the target node of the *rd* link) [2] [3].

Goal model nodes can be also interconnected with directed contribution links. The *++S* link denotes that the satisfaction of source-node contributes positively to the satisfaction of target-node. The *–S* link denotes that the satisfaction of source-node contributes negatively to the satisfaction of target-node. The *++D* link denotes that the negation of the source-node contributes negatively to the satisfaction of target node. Finally, the *–D* link denotes that the negation of the source-node contributes positively to the satisfaction of target node. In addition to these four types of contribution links, Liaskos in [18] proposed *Logical Precedence* links which denote that in order to satisfy the target goal node, the source node must be satisfied first.

In the above model we propose the parallel node, and four new dependency links which extend the expressiveness and the semantics of goal models for not only representing requirements and intentions of agents, but also tasks and actions along with their dependencies. These are:

Parallel Node: It represents a node in the goal model which denotes that the evaluation of two or more nodes(tasks,actions) linked to it can be performed in parallel. In this respect, two or more nodes can be connected via a parallel node form a group of parallelizable nodes. Each Parallel Node is associated with a time limit value denoting the maximum duration in which all the parrallel actions must be performed, otherwise, a timeout event is issued.

Parallel link (par): This type of link connects goal nodes of the model, that must be satisfied in parallel. Many Parallel Links can be connected to the same Parallel Node, creating a group of parallelizable goals.

Temporal Precedence link (tp): It represents a weaker type of relation compared to Logical Precedence (i.e. preconditions) [18], highlighting a temporal relationship between target and source goal node. It denotes that if two actions end up appearing in an execution plan, then the execution of the source action must precede the execution of the target action.

Timeout link (to): It represents the temporal dependency between two Action nodes regarding the maximum period of time between the completion of the one node (i.e. the source node) and the completion of the other node (i.e. the target node) of the *to* link. Otherwise, a timeout event is issued. In this type of link only simple goal nodes (Actions), can be attached and the maximum period of time is provided into the timeout value of the link.

Time Difference link (td): It represents the minimum amount of time that must be passed from the completion of the source Action node, before the execution of the target Action node starts. The amount of time is stored into a field of the link. The edges of this type of connection can only be Action nodes.

An example model and an ADG are illustrated in Figure 1 and in Fig. 2 respectively.

IV. Action Dependency Graph Generation

A. Action Dependency Graphs

An *Action Dependency Graph (ADG)* [2] represents a valid sequence of actions which are ordered in a way that first satisfies the root node of the Goal Model and second, it preserves the temporal dependencies (*tp*, *td*, *to* links), logical dependencies (*lp* links), data dependencies (*rd* links), and contribution dependencies (*++S*, *– S*, *++D*, *–D*) which appear between nodes in the model.

In an ADG, nodes denote *Actions* and edges denote sequential orderings of actions in order to form execution plans. Action nodes in the ADG directly correspond and

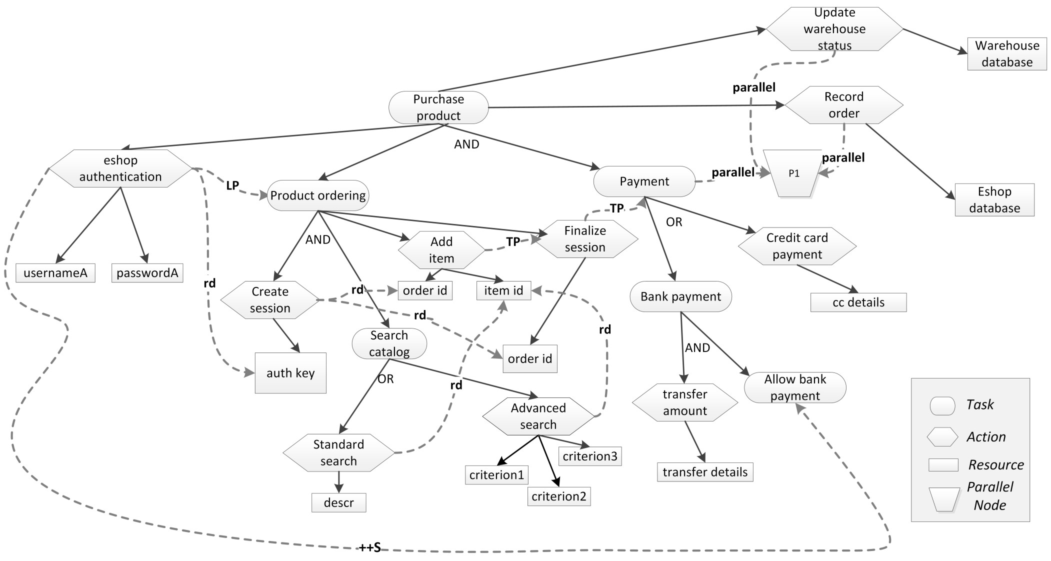

Figure 1: Example of extended Goal Model.

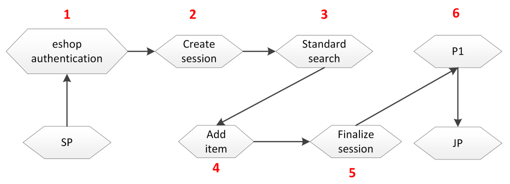

Figure 2: An example topologically sorted ADG for the model in Figure 1.

refer with a 1-1 relations to the actions nodes in the goal model. For example, if a_1 and a_2 are *Action* type of nodes which appear in the goal model, then $a_1 \rightarrow a_2$ in the ADG will denote that in a resulting action plan the execution of action a_1 will precede action a_2.

An example ADG for the model depicted in Fig. 1 is illustrated in Fig. 2. Note here the use of the pseudonode $P1$.

The ADG for a Goal Model is an iterative and incremental process that entails six steps.

Step 1 - line 5: Starting from the bottom of the tree and moving towards the root, one level at a time, we collect all nodes of type *Task* which appear in the Goal Model. The result is a list $\mathcal{T}$ which is ordered based on the depth each task node $t \in \mathcal{T}$ appears in the Goal Model, with the first being the ones with the highest depth.

Step 2 - lines 7-9: For each such task node $t \in \mathcal{T}$, we collect all actions which satisfy the node t. The result is a set $\mathcal{AS}$ of sets s of Action type nodes.

Step 3 - lines 11-17: For each such set $s \in \mathcal{AS}$ we compute a raw "atomic" ADG. The result is a set $\mathcal{R}$ of sets of raw atomic ADGs r. That is there is a raw ADG r for each set $s \in \mathcal{AS}$.

Step 4 - lines 20-25: For each atomic ADG $r \in \mathcal{R}$ we apply the transformation rules discussed in Section IV-B. The role of these transformations is to unravel the parallel nodes, and resolve temporal dependencies, logical dependencies, data dependencies and contribution links at the level of each raw ADG r. The result of the transformations is to yield a processed ADG r' for each raw ADG r.

Step 5 - lines 27-31: On each processed ADG r' we apply topological sorting so that the edges are labeled with numbers in sequence. This sorting will allow for all processed ADGs r' to be fused in sets of final ADGs that *a)* respect the ordering of its constituent ADGs (the r's) and *b)* still satisfy the root node of that segment's root.

Step 6 - line 33: A collection $\mathcal{AP}$ of complete ADGs is computed by considering the possible merges of the individual ADG's r' generated in Step 5. By traversing each topologically sorted ADG in the set $\mathcal{AP}$, a set of individual action plans p can then be formed. Each such action plan p respects all the dependencies of the Goal Model, and also satisfies the root node of that segment. The selection of the optimal plan can be achieved by applying any of the techniques proposed in the research literature as discussed in Section II.

In order to capture the start and end parts of each individual ADG we consider two additional ADG node types. A *Split* node (*SP*) is the root node of the graph and illustrates the beginning of the action sequences that are represented in that graph segment. Moreover, it is the starting point of possible, different threads of actions that must be run in parallel. A *Join* node (*JP*) is the final node of an ADG and represents the end of all action sequences of the graph segment. It is the ending point of possible, different, parallel threads of actions. These two nodes delimit the "start" and "end" points of the individual atomic raw ADGs.

A formal specification of the process is given in Algorithm *Orchestrator* below, while a complete running example of the process is presented in Section IV-C.

Algorithm name: *Orchestrator*
Input: $goalmodel$:$GoalModel$
Output: $actionplans$:[$PlanSet$] where $PlanSet$ is $\{s : s$ is $Action\}$

```
 1: Let $\mathcal{L}$ = []
 2: Let $\mathcal{C} = \emptyset$
 3: Let $\mathcal{T}$ = append([$t$ : Task in the goal model], $root$)
 4: // Order tasks by their depth starting from the bottom of the tree
 5: $\mathcal{T}$ = Order($\mathcal{T}$)
 6: // Find all sets (i.e. combinations) of Actions that satisfy each Task t
    in $\mathcal{T}$
 7: for all $t$ in $\mathcal{T}$ do
 8:     $actSet_t = FindSatActions(t)$[1]
 9: end for
10: // For each set of Actions that satisfy root node create ADG and put
    it in the ADG-Collection $\mathcal{C}$
11: for all $actSet \in actSet_{root}$ do
12:     Let $ADG_k = ConstructRawADG(actSet)$[2]
13:     // Check if ADG is valid
14:     if ContributionRule($ADG_k$) == 1 then
15:         $\mathcal{C} = \mathcal{C} \cup ADG_k$
16:     end if
17: end for
18: $\mathcal{L}$ = append($\mathcal{L}$, $\mathcal{C}$)
19: // Apply the following
20: $\mathcal{L}$ = Parallel Rule($\mathcal{L}$)
21: $\mathcal{L}$ = LogicalPrecedence Rule($\mathcal{L}$)
22: $\mathcal{L}$ = TemporalPrecedence Rule($\mathcal{L}$)
23: $\mathcal{L}$ = ResourceDependency Rule($\mathcal{L}$)
24: $\mathcal{L}$ = Timeout Rule($\mathcal{L}$)
25: $\mathcal{L}$ = Timedifference Rule($\mathcal{L}$)
26: // Do topological sorting to all ADG
27: for all $ADGCol \in \mathcal{L}$ do
28:     for all $ADG$ in $ADGCol$ do
29:         TopologicalSort($ADG$)
30:     end for
31: end for
32: // Combine ADGs between various ADG collections in order to extract
    uniform Action Plans
33: actionPlans = combine($\mathcal{L}$)
34: return actionPlans
```

B. Transformation Rules

Steps 3 and 4 presented above focus on the generation of raw and processed ADGs. These ADGs are generated by a collection of transformations as presented below.

Contribution Link Related Rules: The contribution links (*++S*, *–S*, *++D*, *–D*) dictate the co-occurrence or not of the actions that satisfy the source node with the actions that satisfy the target node in the same action plan. When there is a *–S* contribution type link, we have to ensure that in the final produced action plans, actions that satisfy source node do not coexist with actions that satisfy target node. In the case of *++D* contribution type links, we check that when there is no a set of actions in an action plan that satisfies target node, then there must be no set of actions that satisfies source node. The other two types of contribution links similar dual semantics.

[1]$FindSatActions$ is implemented by transforming the AND/OR decompositions in Conjunctive Normal Form (CNF).

[2]$ConstructRawADG$ is implemented by linking each node in the $actSet$ with a SP node and a JP node.

Parallel Link Rule: For each *Parallel* node that links two or more nodes together, this rule constructs a *collection* of Action Dependency Graph sets ($ADGSet$). Each $ADGSet$ in such a *collection*, contains all the actions that satisfy the nodes (e.g. $Tasks$) that are connected with the this *parallel* node. For example in Figure 3 the *Parallel* node is $P1$ and the connected to it nodes are the task nodes $T2$ and $T3$, which here are specified as being able to be executed in parallel. In this respect, our analysis should not only consider all the different combinations of action sets that satisfy all the task nodes connected to a *Parallel* node, but to also take into account the possible decompositions of the task nodes connected to the *Parallel* node (e.g. $T2$'s and $T3$'s own AND and OR decompositions), so that all possible combinations can be obtained. The parallel link rule creates an ADG pseudo-node which will reference all such action combinations that satisfy the included goals. For each such combination an ADG is constructed and is placed in the place of the pseudo-node. An example of how the parallel rule works for the example model depicted in Figure 3, is illustrated in Figure 5. More formally, the generation of an ADG in the case of a Parallel link rule proceeds as follows:

Algorithm name: *Parallel Rule*
Input: L:[$ADGSet$] where $ADGSet$ is $\{s : s$ is $ADG\}$]
Output: L:[ADGSet] where ADGSet is {s : s is ADG}]

```
 1: Let $\mathcal{P}$ be the set of parallel nodes in the ADG.
 2: for all $P_i \in \mathcal{P}$ do
 3:     // Create an ADG-Collection with name $\mathcal{C}_i$
 4:     $\mathcal{C}_i = \emptyset$
 5:     Let $\mathcal{G}$ be the set of all goal nodes connected to node $P_i$
 6:     // Find the set $\mathcal{S}_{OR}$ of nodes which are OR Decomposed for each
        $g \in \mathcal{G}$
 7:     Let $G_{OR} = \{g \in \mathcal{G} \mid g$ is OR descendant of a node in the
 8:     goal model$\}$
 9:     Let $\mathcal{S}_{OR} = PowerSet(G_{OR})$
10:     // Find the set $\mathcal{S}_{AND}$ of AND Decomposed nodes for each $g \in \mathcal{G}$
11:     Let $S_{AND} = \{\{g \in \mathcal{G} \mid g$ is AND descendant of a node in the
12:     goal model$\}\}$
13:     Let $\mathcal{S}_{all} = \cup_{S \in \mathcal{S}_{OR}} (S \cup S_{AND})$
14:     // Find all combinations of sets of Actions that satisfy the goal
15:     // nodes $g_k$ connected (i.e. $\triangleright$) to parallel node $P_i$
16:     for all $gs \in \mathcal{S}_{all}$ do
17:         Let $acombs = \{ac_k \mid ac_k$ is {a | a:Action} s.t. $ac_k$ satisfies
18:         all goal nodes $g \in gs\}$
19:         for all $ac_k \in acombs$ do
20:             Let $ADG_k$ = Create($ac_k$)
21:             // Check if ADG is valid
22:             if ContributionRule($ADG_k$) ==1 then
23:                 $\mathcal{C}_i = \mathcal{C}_i \cup ADG_k$
```

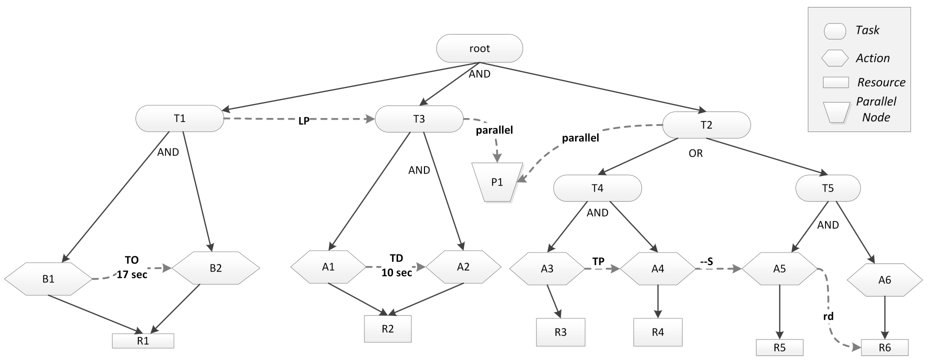

Figure 3: The Goal Model for the running example.

```
24:            end if
25:         end for
26:      end for
27:      $\mathcal{L}$ = append($\mathcal{L}$, $C_i$)
28:  end for
29:  return $L$
```

Logical Precedence Link Rule: The semantics of this link between two nodes imply that a strict sequential ordering must exist between the corresponding actions in the resulting action plans.

Temporal Precedence Link Rule: The connected nodes of this link have a temporal dependency with each other, which implies an execution ordering between the actions that satisfy them. The logic is the same as in Logical Precedence link Rule however, Temporal Precedence expresses a weaker notion of this dependency.

Resource Dependency Link Rule: The target node of this link denotes the data d which the source node x produces. As a result, it is implied an execution order between the source node x of the *rd* link (i.e. the producer) and the parents p of the resource d (i.e the consumer). In this respect, a link $x \overset{rd}{\to} d$ and a decomposition link $p \to d$ will result on a dependency ordering $x \to p$.

Timeout Link Rule: This rule transforms the timeout links of the model into dependencies between action nodes in an ADG, adding also as a constraint the timeout period for the execution of the two actions, which will be taken into account during runtime (see output in Section IV-C).

Time Difference Link Rule: This rule enforces a minimal time constraint between the termination of one action x and the start of the next action y when they appear in an plan. (see sample output in Section IV-C).

C. *Running Example*

A general example of goal model, which we will use as our running example, is illustrated in Figure 3. This model includes parallel nodes, timeout links, resource dependency links, and contribution links.

First, we collect all Tasks in the Goal Model ordered from the ones with the highest depth being first, and including also the $rootnode$ (at the end). We then apply the $FindSatActions$ function (see the *Orchestrator* algorithm above) in order to find the sets of actions that satisfy each *Task* in the Goal Model and we obtain the following results:

- T4:{{A3,A4}}
- T5:{{A5,A6}}
- T2:{{A3,A4},{A5,A6},{A3,A4,A5,A6}}
- T3:{{A1,A2}}
- T1:{{B1,B2}}
- root:{{B1,B2,A1,A2,A3,A4},
  {B1,B2,A1,A2,A5,A6},
  {B1,B2,A1,A2,A3,A4,A5,A6}}

Note that the list of Tasks is sorted according to the depth of each *Task* node in the Goal Model, first being the nodes with the highest depth (see Step 1 & 2 in Section IV-A).

The process proceeds now from the top (i.e. from node $T1$) by creating the initial raw *ADGCollection* which stores all the nodes connected to the root node. The process is not expanding at this point any parallel node (here node P1) (see Step 3 in Section IV-A). Note here the use of the pseudonode P1 which "bundles" nodes T_3 and T_2, as well as the "start" node SP and the "end" node "JP". As a result we will have the ADG depicted in Figure 4.

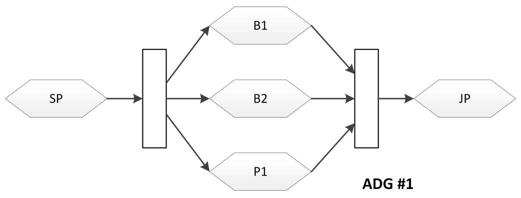

Figure 4: ADG correspondong to nodes at depth 1. Node P1 bundles tasks T2 and T3.

When the algorithm applies the *Parallel Link Rule* (see also Step 4 in Section IV-A), it creates an *ADGCollection* $\mathcal{C}_1$ (see Parallel Rule algorithm) which corresponds to expanding the parallel node $P1$, and which stores the ADGs that satisfy the goals of each set in S_{all} for node $P1$. For our example, the S_{all} set corresponding to the parallel node $P1$ is: {{T2,T3}}.

Then for each set of goals, the algorithm calculates all the possible set of actions that satisfy all the *Task* nodes in each set in S_{all}: {{A1,A2,A3,A4}, {A1,A2,A5,A6}, {{A1,A2,A3,A4,A5,A6}}.

For each one of these sets of actions an ADG is created and consequently checked whether there is a constraint due to contribution links. In this example, we cannot have an ADG that contains the actions A4 and A5 due to the –*S* contribution link between A4 and A5. Note that the –*S* link denotes that the invocation of A_4 prohibits the invocation of A_5. So the set {A1,A2,A3,A4,A5,A6} that contains the actions A4, A5 is eliminated. Finally the following sets kept in ADG Collection $\mathcal{C}_1$ for P_1 are depicted in Figure 5:

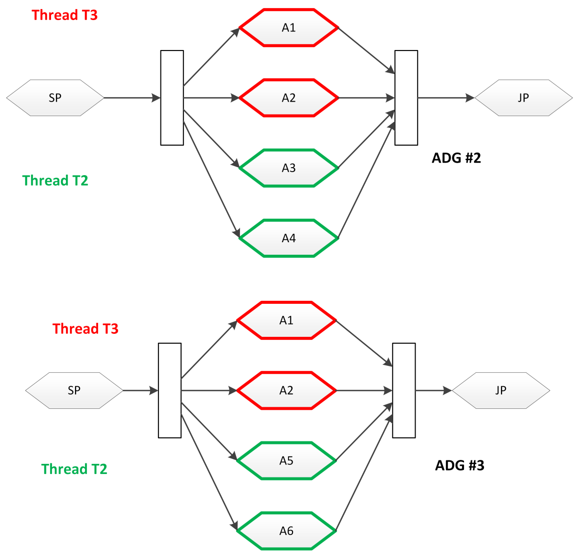

Figure 5: Creation of threads related to node P1.

The next rule which is activated is the $LogicalPrecedence$ rule due to *LP* the link between Task T1 and Task T3 (see also Step 4 in Section IV-A). As a result only ADG #1 will change as B_1 and B_2 have to invoked before the parallel bundle P_1 where Tasks T_3 and T_2 can be performed in parallel and will produce the ordered ADG depicted in Figure 6:

Next, the *TemporalPrecedence Link Rule* is applied on actions on A3 and A4 due to the TP link between A3 and A4. In this respect, A3 and A4 are now ordered in sequence. The resulting ADG is depicted in Figure 7.

Applying the *ResourceDependency Link Rule*, due to the rd link between A5, R6, and A6. The rule imposes an ordering between A5 and A6, as A5 has to first create the resource R6 before A6 consumes it. The ADG that contains them is depicted in Figure 8.

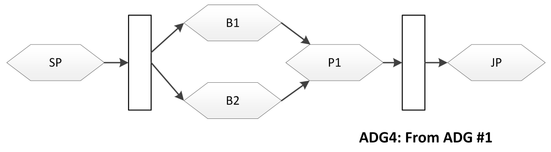

Figure 6: Ordering the bundle for parallel bundle P1 to occur after the actions of task T1.

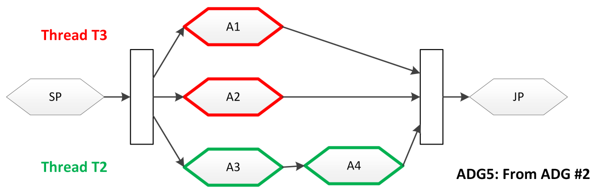

Figure 7: Apply Temporal Precedence between actions A3 and A4.

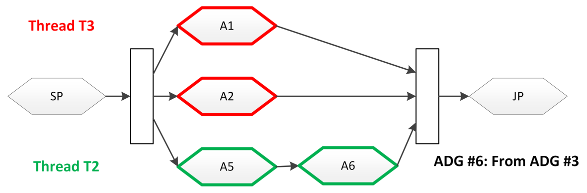

Figure 8: Apply Resource Dependence between actions A5 and A6.

The next rule that is activated is the *TimeOut Link Rule* which defines an execution ordering in the ADG due to the TO link between B_1 and B_2. The ADGs that contain them is depicted in Figure 9:

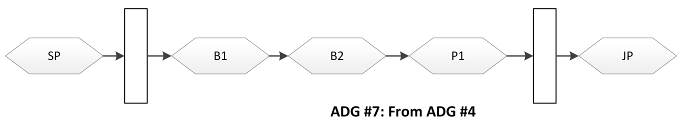

Figure 9: Apply the Timeout Dependence between actions B1 and B2.

The final rule that is triggered is the *TimeDifference Link Rule* due to the TD link between A_1 and A_2 and so the ADGs that contain them is depicted in Figure 10:

Now all the necessary ADGs have been created and a topological sorting is applied to each thread of each ADG. As a result at the end of this procedure we obtain a specific sequence of actions for each ADG. In Figure 11 the ADGs are presented with the final ordering of actions

Finally, the algorithm combines ADGs from the different *ADGCollections* in order to construct the final action plans.

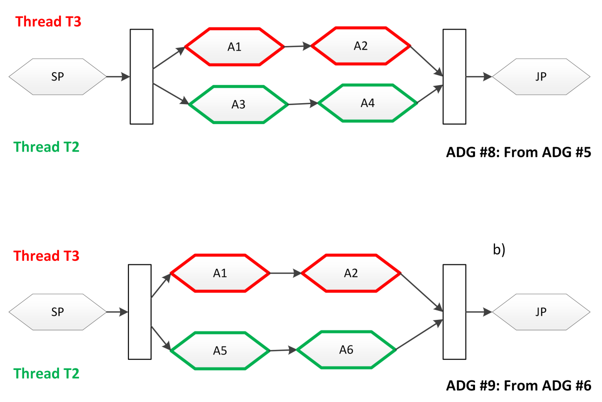

Figure 10: Apply the Time Difference Dependence between actions A1 and A2.

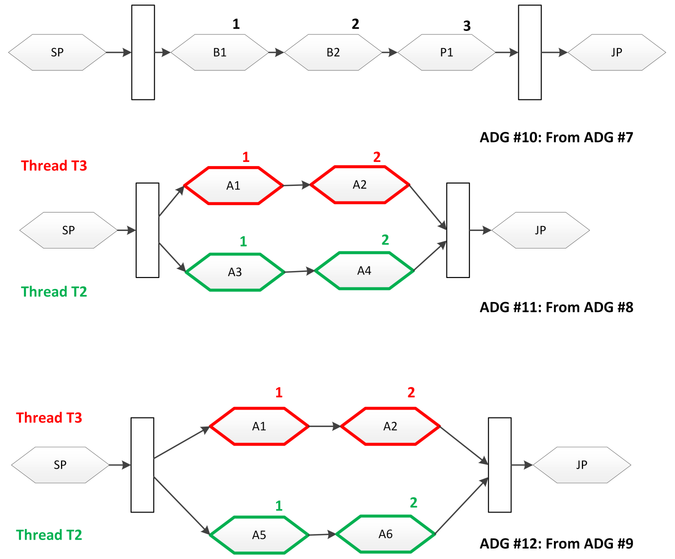

Figure 11: The individual ADGs topologically ordered.

For example combining the ADGs in Figure 11 the resulting action plans are depicted in Figure 12.

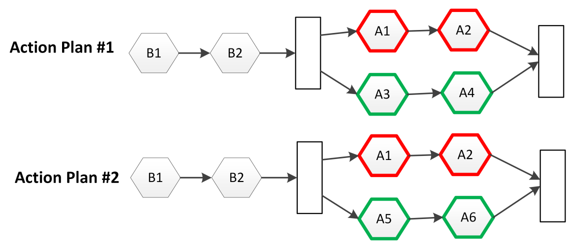

Figure 12: A resulting ordered ADG.

The tool we have implemented enacting the above alorithm provides its output in textual format (JSON, XML, pretty print).

The pretty print output for the running example is:

```
One sequence is:
B1 B2 P1
B2 must complete execution at most 17 secs after B1 has completed its execution

One sequence for parallel P1 is:
thead1 : A1 A2
thread2: A3 A4
A2 must commence execution at least 10 secs after A1 has completed its execution

One sequence for parallel P1 is:
thead1 : A1 A2
thread2: A5 A6
A2 must commence execution at least 10 secs after A1 has completed its execution
```

V. Experiments

Here we present the performance of the algorithm for various goal models of diffferent sizes. The experiments focus on how the number of OR-decompositions affect the performance of the system. Furthermore, the experiments assess how the depth of OR-decomposed nodes and depth of nodes that belong to a parallel group influence time and space performance. For this reason, we define the following variable: $depth = OR\_depth * Parallel\_depth$, where OR_depth denotes the average depth of OR-decomposed nodes and $Parallel\_depth$ denotes the average depth of nodes belonging to a parallel group.

A. Number of OR-Decompositions The number of OR-decompositions is a significant factor of the performance of the system because the number of different action plans that will be produced depends on the different ways the root node can be satisfied. So a large number of OR decompositions can produce a large number of solutions, which may add an overhead to the system.

The experiments have been conducted for different number of nodes and for different number of OR decompositions. During the experiments the parameters of the goal models that are kept constant are *a)* Parallel nodes = 3; *b)* Parallel links = 8; *c)* LogicalPrecedence links = 20; *d)* TemporalPrecedence links = 20; *e)* Contribution links = 8; *f)* Resource Dependency links = 40; *g)* TimeOut links = 20 and; *h)* TimeDifference links = 20.

1) Execution time: Figure 13 depicts the execution time for 5 different numbers of OR-Decompositions.

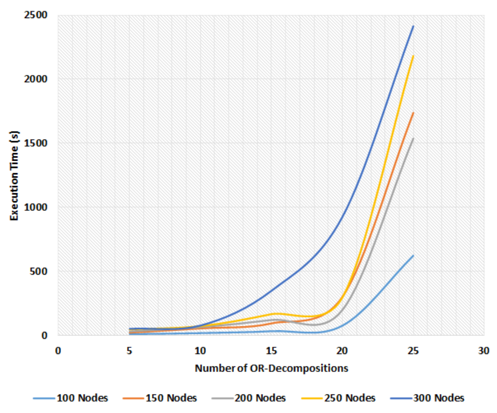

Figure 13: Execution time varying the number of nodes and of OR-Decompositions.

From Figure 13 we can observe that the number of OR-decompositions have an exponential relationship with the execution time of the algorithm. For each OR-decomposition the algorithm must find all these sets of actions that satisfy the child-nodes and then combine them all together in order to produce all the possible sets of actions that satisfy the parent-node. Thus, as expected the behaviour of the algorithm in its current implementation is exponential.

2) Memory Space: For the same goal models, the required memory space is depicted in Figure 14.

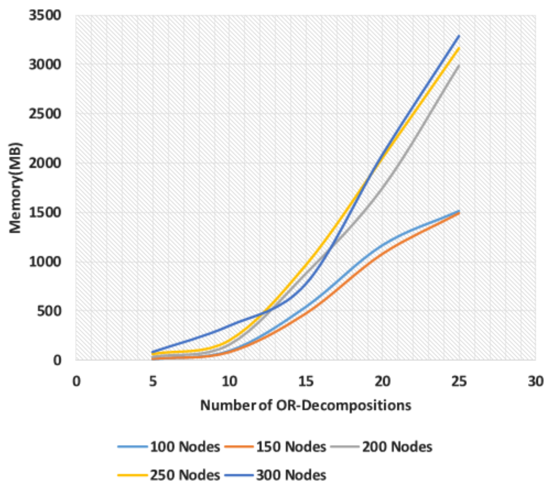

Figure 14: Memory space varying the number of nodes and of OR-Decompositions.

Like in the previous results we can observe an exponential relationship between memory space and number of OR-decompositions for the same reasons as descibed above.

B. Depth The higher an OR-decomposition is located in a goal tree, the more combinations of actions that satisfy the parent-node of the decomposition are produced due to the large subtrees that must be processed below that level of decomposition. Moreover, the depth of a goal node that is connected to a parallel node defines how large will be the subtree that must be processed in order to construct the appropriate ADGs that will be put in the corresponding ADG collection. The size of the subtree and the size of the constructed ADG are proportional and so a high average depth of these goal nodes has a significant impact to the performance of the system. For this reason we define, as explained above, the variable $depth = ORdepth * Paralleldepth$.

The parameters of model we used for experimentation are: *a)* Goal nodes = 150 (Tasks = 100,Actions=50); *b)* LogicalPrecedence links = 20; *c)* TemporalPrecedence links =20; *d)* Contribution links = 8; *e)* Resource Dependency links = 40; *f)* TimeOut links = 20; *g)* TimeDifference links = 20 and; *h)* OR-Decomposition number = 15.

3) Execution time: Figures 15 and 16 depict the execution time for three different configurations of parallel nodes(4, 8 and 12 nodes), along with the corresponding memory requirements.

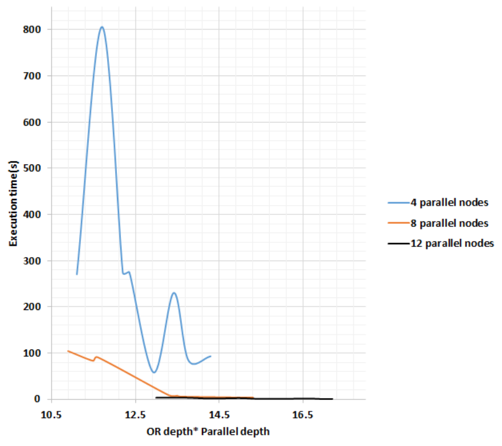

Figure 15: Execution time varying the model height.

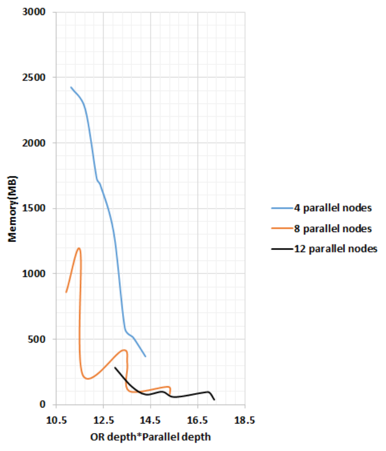

Figure 16: Memory space varying the model height.

From Figures 15 and 16 we can see that the execution time and the requirements in memory are, in general, decreasing while the number of parallel nodes is increasing. For each parallel node the algorithm creates a pseudonode which represents an ADG collection that contains all the goal nodes that are connected to that parallel node. In this respect, the goal model is shattered into a number of constituent pieces that depends on the number of parallel nodes, creating smaller subtrees of the model. In these simpler submodels the algorithm applies the transformation rules, having a better performance in execution time and memory space. The variance in the performance is a consequence of two factors. Firstly, if the various dependency links of the goal model connect, in their majority, *Tasks* and not *Actions*, then the complexity of applying a transformation rule in Tasks is much higher than applying the same rule in Actions. Secondly, despite the fact that we keep constant the average depth of OR-decompositions, we may have a case where an OR-decomposition is located near the root node, creating these observed variations in performance.

VI. Conclusion

This paper presented an incremental graph transformation approach whereby action plans can be composed from Goal Models. More specifically, in this paper we have focused on two main aspects. The first aspect deals with extensions to the Goal Model meta-model, where we are introducing parallelism nodes, and new dependency links related to temporal constraints (*td*, *tp*, *to*) as well as data depenedncies (*rd* and logical dependencies (*ld*) among tasks and actions. The second aspect deals with a graph transformation approach by which extended goal models are incrementally transformed to yield topologically sorted Action Dependency Graphs which produce collections of feasible action plans. These action plans satisfy the underlined Goal Model, and all its dependencies and constraints. These action plans can be directly mapped to existing workflow languages such as BPEL for direct enactment. The proposed approach can be part of a dynamic service provision system where microservices, stakeholders, and infrastructure can be adapted at runtime to fit diverse and possibly conflicting requirements. The proposed approach opens some pointers for future research. First, the transformational approach can be further optimized by introducing parallelism. In this respect, the Goal Model can be decomposed into independent sections, whereby each section can be processed and transformed in parallel with the other sections. Second, each action can be associated with a utility score as a function of cost, performance, reliability, availability, ease of use, and level of trustworthiness. Such an aggregate utility score can then be used to select the optimal action plan among the valid ones. Finally, the proposed technique can be associated with a Model Driven Development approach where infrastructure code such as manifests and service deployment descriptors can be automatically generated from actions plans.

References

[1] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, pp. 249–262, 2001.

[2] G. Chatzikonstantinou, M. Athanasopoulos, and K. Kontogiannis, "Towards a goal driven task personalization specification framework," in *2013 IEEE Ninth World Congress on Services*, pp. 180–184, IEEE, 2013.

[3] G. Chatzikonstantinou, M. Athanasopoulos, and K. Kontogiannis, "Task specification and reasoning in dynamically altered contexts," in *International Conference on Advanced Information Systems Engineering*, pp. 625–639, Springer, 2014.

[4] T. Eijndhoven, M.-E. Iacob, L. Ponisio, and a. Laura, "Achieving business process flexibility with business rules," pp. 95–104, 10 2008.

[5] I. Markovic and M. Kowalkiewicz, "Linking business goals to process models in semantic business process modeling," in *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pp. 332–338, 2008.

[6] G. Koliadis, "Relating business process models to goal-oriented requirements models in "kaos"," in *PKAW - LNAI 4303*, p. 25–39, 2006.

[7] F. Mardukhi, N. Nematbakhsh, K. Zamanifar, and A. Barati, "Qos decomposition for service composition using genetic algorithm," *Applied Soft Computing*, vol. 13, p. 3409–3421, 07 2013.

[8] H. Wang, X. Hu, Q. Yu, M. Gu, W. Zhao, J. Yan, and T. Hong, "Integrating reinforcement learning and skyline computing for adaptive service composition," *Information Sciences*, vol. 519, pp. 141 – 160, 2020.

[9] H. Zo, D. L. Nazareth, and H. K. Jain, "Service-oriented application composition with evolutionary heuristics and multiple criteria," *ACM Transactions on Management Information Systems (TMIS)*, vol. 10, pp. 1 – 28, 2019.

[10] R. Groenmo and M. C. Jaeger, "Model-driven semantic web service composition," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pp. 8 pp.–, 2005.

[11] K. Rim, C. Souveyet, and C. Rolland, "Eliciting service composition in a goal driven manner," pp. 308–315, 11 2004.

[12] E. Khanfir, R. B. Djmeaa, and I. Amous, "Self-adaptive goal-driven web service composition based on context and qos," in *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, pp. 201–207, 2017.

[13] B. Orriens, J. Yang, and M. Papazoglou, "A framework for business rule driven web service composition," vol. 2819, pp. 52–64, 09 2003.

[14] M. Jiang and X. Zhang, "A requirement-driven web service composition modeling framework," in *2015 International Conference on Computer Science and Applications (CSA)*, pp. 293–297, 2015.

[15] E. S. Yu and J. Mylopoulos, "Understanding" why" in software process modelling, analysis, and design," in *Proceedings of 16th International Conference on Ssoftware Engineering*, pp. 159–168, IEEE, 1994.

[16] J. Mylopoulos, L. Chung, and E. Yu, "From object-oriented to goal-oriented requirements analysis," *Communications of the ACM*, vol. 42, no. 1, pp. 31–37, 1999.

[17] A. Dardenne, S. Fickas, and A. van Lamsweerde, "Goal-directed concept acquisition in requirements elicitation," in *Proceedings of the 6th International Workshop on Software Specification and Design*, pp. 14–21, IEEE Computer Society Press, 1991.

[18] S. Liaskos and J. Mylopoulos, "On temporally annotating goal models.," in *IStar*, pp. 62–66, 2010.