Exploiting the hierarchical structure of rule-based specifications for decision planning

Artur Boronat[1], Roberto Bruni[2], Alberto Lluch Lafuente[3], Ugo Montanari[2], and Generoso Paolillo[4]

[1] Department of Computer Science, University of Leicester, UK
[2] Department of Computer Science, University of Pisa, Italy
[3] IMT Institute for Advanced Studies Lucca, Italy
[4] Laboratorio CINI-ITEM Carlo Savy, Naples, Italy

Abstract. Rule-based specifications have been very successful as a declarative approach in many domains, due to the handy yet solid foundations offered by rule-based machineries like term and graph rewriting. Realistic problems, however, call for suitable techniques to guarantee scalability. For instance, many domains exhibit a hierarchical structure that can be exploited conveniently. This is particularly evident for composition associations of models. We propose an explicit representation of such structured models and a methodology that exploits it for the description and analysis of model- and rule-based systems. The approach is presented in the framework of rewriting logic and its efficient implementation in the rewrite engine Maude and is illustrated with a case study.

1 Introduction

Rule-based specifications have been very successful as a declarative approach in many domains. Prominent examples from the software engineering field are architectural reconfiguration, model transformation and software refactoring. One of the key success factors are the solid foundations offered by rule-based machineries like term and graph rewriting. Still, the complexity of realistic problems requires suitable techniques to guarantee the scalability of rule-based approaches. Indeed, the high number of entities involved in realistic problems and the inherently non-deterministic nature of rule-based specifications leads to large state spaces, which are often intractable.

Fortunately, many domains exhibit an inherently hierarchical structure that can be exploited conveniently. We mention among others nested components in software architectures, nested sessions and transactions in business processes, nested membranes in computational biology, and so on. In this paper we focus on the structure of *model-based* specifications due to various motivations. First, it is widely accepted that *models* enhance software comprehension [19]. Second, many model-driven development and analysis activities demand efficient and scalable approaches. Our approach aims at enhancing software comprehension by making explicit some of the structure of models, and at improving rule-based analysis techniques by exploiting such structure. For instance, the Meta-Object

Facility (MOF) standard defines a metamodelling paradigm by providing a set of UML-like structural modelling primitives including composition associations. Such associations impose a hierarchical structure on models. However, models are usually formalised as flat configurations (e.g. graphs) and their manipulation is studied with tools and techniques based on term rewriting or graph transformation theories [7] that do not exploit the hierarchical structure. For instance, in the MOF, models are collections of objects that may refer to other objects through references, corresponding to flat graphs in the traditional sense. In addition, some of these references are typed with composition associations in a metamodel and their semantics corresponds to structural containment. In this way, models have an *implicit* nested structure since some objects may contain other objects. To the best of our knowledge, a formalism with an *explicit* notion of structural containment has not been used for specifying model-based software artefacts yet.

In this paper we propose a formal representation of models that makes explicit the hierarchical structure of containment and a methodology that exploits such information for the description and analysis of model- and rule-based systems. The main class of analysis we shall address in this paper are planning problems that arise in various engineering activities that rely on rule-based declarations, like devising architectural reconfiguration plans, executing model transformations or taking refactoring decisions. Such problems have particular characteristics that make them different from traditional approaches. First, states in traditional planning tend to be *flat*, i.e. they typically consist of sets of ground predicates. Instead, our states are *structured* models represented by terms, offering rich descriptions that we would like to exploit conveniently. Second, rules in traditional planning are typically first-order and application conditions do not include rewrites but are limited to state predicates. Instead, our rules are *conditional* term rewrite rules à la Meseguer [14], i.e. variables can be bound to subterms and conditions can be rewrite rules whose results are used in the right-hand side. Such rules are needed to exploit structural induction during model manipulations. Third, our rules are decorated with *labels* that are used to both coordinate and guide the manipulation of models, in the spirit of *Structural Operational Semantics* [16] (SOS) and its implementation in rewriting logic [20]. We believe that the success of this discipline in the field of language semantics can be exported to model-driven transformations. Fourth, we consider *multi-criteria* optimisation where several dimensions can be used to find non-dominated optimal or near-to-optimal solutions and our approach is independent on the actual choice of quantitative aspects. This is achieved by using a generic algebraic, compositional formalism for modelling quantitative criteria, namely some variant of semirings [2], and devising plan optimisation methods that are valid for any semiring. In that way we can measure and accordingly select the most convenient model manipulations when various choices are possible, e.g. architectural reconfigurations ensuring a good load-balance but involving a low number of re-bindings, or class diagram refactorings reducing the number of classes but not requiring too many method pull-ups.

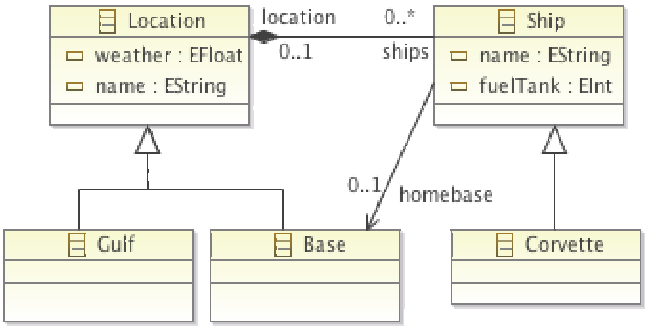

Fig. 1. Class diagram for the navy missions scenario

For this purpose we define some basic machinery based on rewriting logic and we devise a methodology to use it in practice with Maude [5], the rewrite engine tool supporting rewriting logic. In particular this paper presents 1) a novel representation of models based on nested collections of objects described with rewriting logic, 2) a methodology for exploiting the nesting structure in the declaration of rules, 3) a purely declarative presentation of planning problems with multi-criteria optimisation, i.e. we do not implement any new algorithm in Maude, but rely on Maude's reachability capabilities.

Synopsis. § 2 describes a running example, based on an industrial case study. § 3 summarises the mathematical machinery we rely on. § 4 presents the core fundamentals of our approach. § 5 explains how problem domains and instances are described and analysed. § 6 discusses related work. § 7 concludes the paper and outlines future research avenues.

2 Running Example: Navy Missions Scenario

Our running example is a naval scenario taken from a case study developed in a collaboration with the Italian company *Selex Sistemi Integrati* within the national project TOCAI.IT.[5] Basically, it consists of a decision support system to integrated logistic during dynamic planning of navy operations. The considered scenario consists of a naval fleet that while carrying out its current mission, is then required to switch mode of operation because some unpredictable events happened that impose new objectives with higher priority. For example, a patrol activity for peacekeeping along a coast can be required to switch to a rescue activity of civil population after a natural disaster. The re-planning requires the modelling of the new objectives and constraints that characterize the new

[5] http://www.selex-si.com; http://www.dis.uniroma1.it/~tocai.

mission and the subsequent evaluation of feasible and most convenient logistic action plan to be exploited for achieving the new goal.

Figure 1 depicts a simplified excerpt[6] of the class diagram for our running example, including only ingredients that we shall use throughout the paper. In particular, we see the classes for locations (Location) and ships (Ship). Two particular subclasses of locations are distinguished (Base and Gulf) as well as a particular subclass of ship (Corvette). Locations can contain ships. Ships can have a reference to their home base (homebase). Most of the classes have attributes, like a name for ships and locations (name), the fuel remaining in the tank of a ship (fuelTank) or the weather conditions for a location (weather).

3 Technical background

Rewriting Logic. Our specifications are theories described by rewriting logic [14].

Definition 1 (rewrite theory). *A* rewrite theory $\mathcal{R}$ *is a tuple* $\langle \Sigma, E, R \rangle$ *where* Σ *is a signature, specifying the basic syntax (function symbols) and type machinery (sorts, kinds and subsorting) for terms, e.g. model descriptions;* E *is a set of (possibly conditional) equations, which induce equivalence classes of terms, and (possibly conditional) membership predicates, which refine the typing information;* R *is a set of (possibly conditional) rules, e.g. actions.*

The signature Σ and the equations E of a rewrite theory form a *membership equational theory* $\langle \Sigma, E \rangle$, whose initial algebra is $T_{\Sigma/E}$. Indeed, $T_{\Sigma/E}$ is the state space of a rewrite theory, i.e. states are equivalence classes of Σ-terms (denoted by $[t]_E$ or t for short). Usually, one is not interested in considering any term to be a state: for instance, a term can represent a part of a model like the attributes of an object. In such cases, a designated sort `State` is used and the state space of interest is then $T_{\Sigma/E,\texttt{State}}$, i.e. all `State`-typed terms (modulo E).

Rewrite rules in rewriting logic are of the form $t \to t'$ if c, where t, t' are Σ-terms, and c is an application condition (a predicate on the terms involved in the rewrite, further rewrites whose result can be reused, memberships, etc.).

Semirings. Our specifications will be equipped with *quantitative information* such as the value of attributes or non-functional properties associated to rules. For instance, in our case study we are interested in modelling duration and risk factor of actions. There are many heterogeneous notions of quantitative features such as probabilistic, stochastic or time-related aspects, and for each one, specialised formalisms capturing their essence, e.g. Markovian models. Instead of a very specialised model, we use a generic, flexible framework for the representation of quantitative information. More precisely, we consider *semirings*, algebraic structures that have been shown to be very useful in various domains, notably in *Soft Constraint Problems* [2]. The main idea is that a semiring has a domain of

[6] The full scenario contains further entities, inheritance relations, and composition associations like fleets being made of ships, ships containing crafts, and so on.

partially ordered values and two operations: one for choosing the best between two values (a greatest lower bound), and another one for combining values. We focus on a particular variant of semirings, namely *constraint-semirings* (semirings, for short).

Definition 2 (semiring). *A* semiring *is a tuple* $\langle A, \sqcup, \otimes, \mathbf{0}, \mathbf{1} \rangle$ *such that* A *is a (partially ordered) set of values;* $\mathbf{0}$ *and* $\mathbf{1}$ *are the bottom (worst) and top (best) values of* A*;* $\sqcup : A \times A \to A$ *is an operation to choose values: it is associative, commutative, and idempotent, has* $\mathbf{0}$ *as its unit element and* $\mathbf{1}$ *as its absorbing element;* $\otimes : A \times A \to A$ *is an operation to combine values: it is associative, commutative, distributes over* $\sqcup$*, has* $\mathbf{1}$ *as its unit element and* $\mathbf{0}$ *as its absorbing element. The choice operation coincides with the join operation of the lattice induced by* $a \sqsubseteq b$ *iff* $a \sqcup b = b$*.*

Notable examples are the *Boolean* ($\langle \{true, false\}, \vee, \wedge, false, true \rangle$), the *tropical* ($\langle \mathbb{R}^+, min, +, +\infty, 0 \rangle$), the *max/min* ($\langle \mathbb{R}^+, max, min, 0, +\infty \rangle$), the *probabilistic* ($\langle [0,1], max, \cdot, 0, 1 \rangle$), the *fuzzy* ($\langle [0,1], max, min, 0, 1 \rangle$), and the *set* ($\langle 2^N, \cup, \cap, \emptyset, N \rangle$) semirings. For instance, action duration is modelled in our case study with a tropical semiring. In that way, time is modelled as a positive real value, choosing between two actions means choosing the fastest one and combining two actions means adding their durations (i.e. combining them sequentially). Similarly, the risk factor is modelled with a fuzzy semiring.

Semiring based methods have a unique advantage when problems with multiple QoS criteria must be tackled: Cartesian products, exponentials and power constructions of semirings are semirings. Thus the same concepts and algorithms can be applied again and again. For instance, given two semirings C_1 and C_2 their Cartesian product $C_1 \times C_2$ is a semiring. This allows us to deal with multiple criteria at once. Moreover, such meta-operations can be implemented using Maude's parameterized modules and module operations. For example, the quantitative information regarding duration and risk of actions in our case study is modelled by the Cartesian product of the corresponding semirings.

Transition systems. The semantics of our rewrite theories are a sort of quantitative transition systems (inspired by [13]) based on the ordinary one-step semantics of rewrite theories [5].

Definition 3 (transition system). *A* quantitative transition system *is a tuple* $\langle S, \Longrightarrow, C \rangle$ *such that* S *is a set of (system) states;* C *is a semiring* $\langle A, \sqcup, \otimes, \mathbf{0}, \mathbf{1} \rangle$ *modelling the quantitative information of the system;* $\Longrightarrow \subseteq S \times A \times S$ *is a transition relation.*

We shall denote a transition (s, q, s') by $s \Longrightarrow_q s'$. We restrict our attention to finitely branching transition systems (i.e. $\forall s \in S. |\{(s, a, s') \in \Longrightarrow\}|$ is finite). The *runs of a transition system* are the (possibly infinite) paths in the underlying state transition graph, i.e. sequences $s_0 \Longrightarrow_{q_0} s_1 \Longrightarrow_{q_1} \ldots$. A finite run $s_0 \Longrightarrow_{q_0} s_1 \Longrightarrow_{q_1} \ldots s_n$ will be denoted by $s_0 \Longrightarrow^*_{\bigotimes q_i} s_n$.

Planning problems. Finally, we formalise some classic planning problems, remarking that many model-driven engineering activities like reconfiguration, refactoring or transformations can be understood as plannning problems.

Definition 4 (planning problem). *Let $T = \langle S, \Longrightarrow, C\rangle$ be a transition system, $I \subseteq S$ be a set of* initial *states and $G \subseteq S$ be a set of* goal *states (typically characterised with predicates). A* planning problem *is given by the tuple $\langle T, I, G\rangle$. A* solution *to a planning problem $\langle T, I, G\rangle$ is a run $s \Longrightarrow_q^* s'$, such that $s \in I$ and $s' \in G$. An* optimal *or* non-dominated solution *is a solution $s \Longrightarrow_q^* s' \in G$ such that there is no other solution $s_1 \Longrightarrow_{q'}^* s_1' \in G$ such that $q \sqsubseteq q'$.*

4 A formalism for structured model- and rule-based specifications

We present the formal means for describing model- and rule-based specifications based on the machinery of §3.

Models as nested objects. In our view, a model is a collection of possibly hierarchical objects, i.e. an object of the system may itself be a complex sub-system composed by various nested objects. The description of models is done with a signature of *nested objects* that extends Maude's object-based signature [5] with nesting features that allow for objects to contain object collections.

More precisely, our rewrite theories are based on a basic membership equational theory $\mathcal{M}_\mathcal{N} = \{\Sigma_\mathcal{N}, E_\mathcal{N}\}$ that provides the main signature and equations. Signature $\Sigma_\mathcal{N}$ is basically made of sorts $K_\mathcal{N}$ and operator symbols $O_\mathcal{N}$.

Definition 5 (basic sorts). *The set of basic sorts of $K_\mathcal{N}$ contains* `Conf`*, i.e. the sort of model configurations;* `Obj`*, i.e. the sort of objects;* `Att`*, i.e. the sort of attributes; a sort* $\texttt{Set}\{T\}$ *for each of the above sorts T, i.e. the sort of sets of T-terms;* `Oid` *of object identifiers; sort* `Cid` *of object classes.*

Sort `Conf` will be our designated `State` sort whenever we will be interested in analysing the space of possible system model configurations. We define now the symbols of the operators that build terms of the above defined sorts.

Definition 6 (basic operators). *The set of basic operator symbols $O_\mathcal{N}$ contains a constructor* $[\,\cdot\,] : \texttt{Set}\{\texttt{Obj}\} \to \texttt{Conf}$ *for configurations, given a set of objects; a constructor* $< \cdot : \cdot \mid \cdot \mid \cdot > : \texttt{Oid} \times \texttt{Cid} \times \texttt{Set}\{\texttt{Att}\} \times \texttt{Set}\{\texttt{Obj}\} \to \texttt{Obj}$ *for objects, such that <o:c|a|s> is an object with identity o, class c, attribute set a and sub-objects s; a constant* $\texttt{none} : \to \texttt{Set}\{T\}$ *for each sort* $\texttt{Set}\{T\}$*, i.e. the empty set; a binary operator* $\cdot\,,\,\cdot : \texttt{Set}\{T\} \times \texttt{Set}\{T\} \to \texttt{Set}\{T\}$ *for each sort* $\texttt{Set}\{T\}$*, i.e. set union.*

Attribute and identifier constructors are problem-dependent, i.e. they are defined for the particular domain or instance being described. We just remark that they typically take the form $n{:}v$, where n is the attribute name and v is the attribute value. Usual attributes include references to object identifiers and quantitative information (see §3).

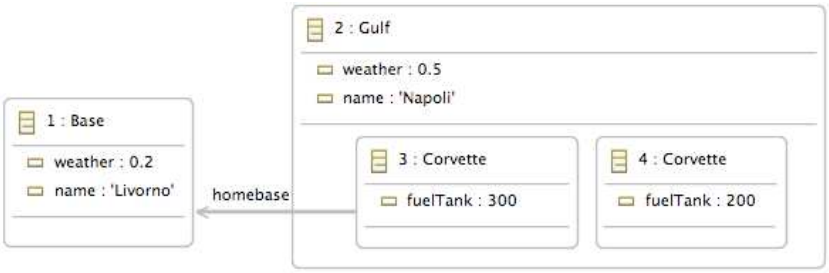

Fig. 2. A configuration with nested objects

Example 1. Consider the diagram of Fig. 2, which illustrates an instance of our scenario where two corvettes are located at a gulf. More precisely, we see that the gulf is represented by the object of class `Gulf` with identifier `2`, various attributes and embedding the vessels within its area, namely objects `3` and `4`, both of class `Corvette`. Instead, Object `1` is the `Base` to which the corvette named `Cassiopea` refers as its home, by means of the reference attribute `homebase` $:\cdot$: `Oid` $\rightarrow$ `Att`. In our notation the described scenario is denoted by term

$$
\begin{array}{l}
[< \texttt{1} : \texttt{Base} \mid \texttt{name} : \texttt{Livorno} \ , \ \texttt{weather} : \texttt{0.2} \mid \texttt{none} > \ , \\
\ < \texttt{2} : \texttt{Gulf} \mid \texttt{name} : \texttt{Napoli} \ , \ \texttt{weather} : \texttt{0.5} \mid \\
\qquad < \texttt{3} : \texttt{Corvette} \mid \texttt{name} : \texttt{Galileo} \mid \texttt{none} > \ , \\
\qquad < \texttt{4} : \texttt{Corvette} \mid \texttt{name} : \texttt{Cassiopea} \ , \ \texttt{homebase} : \texttt{1} \mid \texttt{none} >>]
\end{array}
$$

The set of equations $E_{\mathcal{N}}$ of our basic membership theory essentially axiomatises sets, i.e. it contains equations to denote the associativity, commutativity and idempotency of set union and the fact that the empty set is the identity element for set union.

Definition 7 (basic equations). *The set of basic operator symbols $E_{\mathcal{N}}$ contains equations x , $\texttt{none} = x$ (identity); x , $x = x$ (idempotency); x , $y = y$, x (commutativity); x , $(y$, $z) = (x$, $y)$, z (associativity) for each sort $\texttt{Set}\{T\}$, with $x, y, z : \texttt{Set}\{T\}$.*

Obviously, the designer might introduce new sorts, subsorting declarations or derived operators (new symbols and appropriate equations) for its own convenience, but the above presented signature is at the core of all specifications.[7]

Quantitative information. Semirings can be described with membership equational theories.

[7] Our incremental presentation does not only facilitate the reading of the paper but is supported by module importation in Maude which also has a mathematical meaning in rewriting logic, e.g. $\texttt{Set}\{T\}$ is a parametric module in Maude offering the mentioned constructors and equations.

Definition 8 (semiring theory). *A* semiring theory *is a membership equational theory* $\langle \Sigma, E \rangle$ *such that* Σ *contains the sort* `Cost` *for carrier* A*, the semiring operator symbols* $\sqcup, \otimes, \boldsymbol{0}, \boldsymbol{1}$*, the sort of Booleans and the usual lattice operator symbols.* E *contains the axioms of Def. 2 and the usual equations for lattices.*

A *concrete semiring theory* $\langle A, \sqcup, \otimes, \mathbf{0}, \mathbf{1} \rangle$ is membership equational theory that can be declared as a view of the semiring theory. For instance, The Boolean semiring theory is a view of the usual Boolean theory. This allows us to re-use Maude's predefined theories (e.g. Floating-point numbers as approximation of reals).

Conditional, labelled, quantitative rules. We are interested in rules in a particular format, namely in the style of *Structural Operational Semantics* [16] (SOS). SOS rules guarantee us a firm discipline in specifying the dynamics of a model by structural induction, i.e. by composing the transition of objects exploiting the structure of the model. In addition, we are interested in rules carrying quantitative information. More precisely, one of the rule formats the designer should follow is[8]

$$\frac{t_1 \xrightarrow{l_1}_{q_1} t_1' \qquad t_2 \xrightarrow{l_2}_{q_1} t_2'}{t_1 \ , \ t_2 \xrightarrow{l_1 \odot l_2}_{q_1 \oplus q_2} t_1' \ , \ t_2'} \text{ if } c$$

where a transition for a configuration made of sub-parts t_1 and t_2 (of sort `Set{Obj}`) is inferred from the transitions of each of the parts. More precisely, if part i of the configuration is in state t_i and is ready to perform an l_i-labelled transition to go into state t_i' with cost q_i, then a model configuration with state $t_1 \ , \ t_2$ is ready to perform a transition labelled by $l_1 \odot l_2$ to go into state $t_1' \ , \ t_2'$ with cost $q_1 \oplus q_2$. Eventually, some additional conditions c might be considered, but we require them to be predicates and not additional rewrites.

Operations $\odot$ and $\oplus$ are used to combine transition labels and costs, respectively. Typically, the combination of labels will follow some classical form. For instance, in synchronisation rules, l_1 and l_2 can be complementary actions, in which case $l_1 \odot l_2$ would be a silent action label τ. However, we will not make any particular choice of the synchronisation algebra. It is up to the system designer to decide which labels and label synchronisation to apply. We only remark that it is also possible to use semirings to model classical synchronisation algebras [12]. In the following we assume that our basic signature is enriched with a sort `Lab` for action labels. As for the quantitative information, the actual choice of operation $\oplus$ in each rule is up to the system designer.

In addition, a designer will be allowed to denote the transition of an object, provided that its sub-objects t are able to perform some transition:

$$\frac{t \xrightarrow{l}_{q} t'}{\langle o{:}C | A | t \rangle \xrightarrow{l \odot l'}_{q \oplus q'} \langle o{:}C | A' | t' \rangle} \text{ if } c$$

[8] Note that we put the rule conclusion on the bottom, the rewrite premises on top and additional conditions on a side to stick to the usual SOS notation.

Such rules might affect the attributes of the container object and manipulate the action label but of course are not allowed to change the object's identifier or class. More elaborated versions of the above rule are also allowed, for instance involving more than one object or not requiring any rewrite of contained objects. We shall see some examples in §5.

Finally, there is a rule that is common to all specifications which allow us to derive a global step of a configuration made of a set of objects t, removing the action label, but keeping the transition cost:

$$\frac{t \xrightarrow{l}_q t'}{[t] \longrightarrow_q [t']}$$

Rewrite rules in rewriting logic are not equipped with quantitative information and that rule labels can be used only at the meta-level. This is not a problem, as there are standard techniques to encode transition annotations into states. In particular, we follow the encoding of SOS semantics in rewriting logic [20] and enrich our signature with sorts for action-prefixed states ($\texttt{Act\{State\}}$), a constructor $\{\cdot,\cdot\}\cdot : \texttt{Lab} \times \texttt{Cost} \times \texttt{State} \to \texttt{Act\{State\}}$ for action and cost prefixed states and a constructor $\{\cdot\}\cdot : \texttt{Cost} \times \texttt{State} \to \texttt{Act\{State\}}$ for cost prefixed states. In addition, we enforce rule application at the top-level of terms only (via Maude's `frozen` attribute) so that sub-terms are rewritten only when required in the premise of a rule (as required by the semantics of SOS rules). However, since this is basically an implementation issue, in the rest of the paper we shall continue using our notation of labelled, cost-annotated transitions, leaving implicit the fact that a rewrite $t \xrightarrow{l}_q t'$ actually denotes a rewrite $t \longrightarrow \{l,q\}t'$. In other words, quantitative information is conceptually associated to transitions, but the actual rewriting logic description constrains us to associate it to states, i.e. to use terms to represent that "a state t' was reached with cost q via an l-labelled rule".

Transition system for planning. The built-in tools of Maude allows us to explore the state space of rewrite theories. Basically, we concentrate on Maude's reachability analysis which allows us to find a rewrite sequence from a term t to a term t' satisfying some conditions. To be able to use such tools, we have to encode the analysis problems raised in §3 as rewrite theories. First, we remark that the one-step semantics of the kind of rewrite theories we are interested in is defined as follows.

Definition 9 (one-step semantics). *Let $\mathcal{R}$ be a rewrite theory equipped with a semiring C and with a designated state sort* `State`*. The transition system associated to $\mathcal{R}$ is $T(\mathcal{R}) = \langle S, \Longrightarrow, C\rangle$ such that $S = T_{\Sigma/E,\texttt{State}}$, i.e. states are equivalence classes of terms of sort* `State`*; $\Longrightarrow = \{t \Longrightarrow_q t' \mid t \longrightarrow_q t'$ is a one-step rewrite proof in $\mathcal{R}$ and t,t' are* `State`*-typed terms$\}$, i.e. system transitions are formed by one-step rewrites between states.*

Non-optimal Planning. Now, we concentrate on finding a solution to a planning problem $\langle T, I, G\rangle$ where T is the transition system of the rewrite theory $\mathcal{R}$ describing our specification, I is a set of initial configurations and G is the set of goal configurations. This can be done using Maude's search capabilities. However, the presence of quantitative information introduces unnecessary redundancy. Indeed, the state space might contain duplicate states with different cost annotations. Therefore, we can forget the quantitative information just by dropping the cost annotation. Technically, this is achieved in an elegant way by introducing in $\mathcal{R}$ the equation $\{q\}t = \{\mathbf{1}\}t$. It is obvious to see that reachability in the resulting theory is enough for finding a solution of the planning problem.

Optimal Planning. Now we explain how to find optimal solutions to a planning problem via Maude's reachability analysis. The main idea is to emulate Dijkstra's shortest path algorithm, by exploring the state space of sets of non-dominated configurations. We enrich state annotations with information to explictly record the path to a state: $\{q, p\}t$ denotes that state t has been reached through path p with cost q. In addition, we use path operations like path concatenation (denoted with $\cdot$). Now, we let `Set{Conf}` be the designated sort `State` and we enrich our rewrite theory with the following rule

$$\frac{t \longrightarrow_{q'} t'}{\{q,p\}t \ , \ S \longrightarrow \{q,p\}t \ , \ \{q \otimes q', p \cdot t \Longrightarrow_{q'} t'\}t' \ , \ S} \text{ if } c$$

where c forbids the new state t' to be a dominated duplicate ($\neg\exists\{q'', p''\}t'' \in (\{q,p\}t \ , \ S) \mid t' = t'' \wedge \{q \otimes q', p \cdot t \Longrightarrow_{q'} t'\}t' \sqsubseteq \{q'', p''\}t''$), and the state t selected for exploration to be dominated ($\neg\exists\{q'', p''\}t'' \in S \mid \{q,p\}t \sqsubseteq \{q'', p''\}t''$). The rule basically allows us to enrich the set of discovered configurations so far in a monotonic way. Of course, we have to discard dominated configurations by introducing equation $\{q_1\}t \ , \ \{q_2\}t = \{q_1\}t$ if $q_2 \sqsubseteq q_1$.

We denote the resulting rewrite theory by $\mathcal{R}^{\mathtt{Set}}$. This provides us with a simple method for finding optimal solutions to the planning problem, as each rewrite step roughly emulates an iteration of Dijkstra's shortest-path algorithm (which can be generalised to semirings [17]).

Proposition 1 (optimal planning correctness). *Let $\mathcal{R}$ be a rewrite theory describing a system, I be the set of initial states and G be the set of goal states. Then a solution for the planning problem $\langle T(\mathcal{R}^{\mathtt{Set}}), I, G'\rangle$ is an optimal solution for the planning problem $\langle T(\mathcal{R}), I, G\rangle$, where $G' = \{S' \subseteq S \mid S' \cap G \neq \emptyset\}$.*

5 Domain and instance specification

This section provides some hints for describing and analysing model- and rule-based specifications in our methodology.

System domain description. System descriptions must include actual object classes and attributes. For each class C the designer must declare a sort C that represents the class and a constructor $C :\rightarrow C$. Each sort C is declared as a subsort of `Cid`. Additional subsorting relations might be added in the same spirit of class inheritance. The subsorting of classes allows us to declare rules that apply to certain classes of objects only, in a very convenient way. For instance, in our case study we have classes for the different entities involved in the scenario, like locations (`Location`, `Gulf`, `Base`) and ships (`Ship`, `Corvette`). Sorts `Gulf` and `Base` are declared as subsorts of `Location`, and similarly for `Corvette` and `Ship`.

Next, attribute domains and constructors must be declared. Typically, attributes take the form $n{:}v$, where n is the attribute name and v is the attribute value. Typical attributes include references to object identifiers and quantitative information. In our example, for instance, we use an attribute `homebase` with domain `Oid` to allow for ships to refer to their home bases and we have an attribute `weather` with a fuzzy semiring as domain to represent the risk factor introduced by weather conditions.

Of course, the system designer might introduce additional sorts, subsorting declarations or operators (new symbols and appropriate equations) for his own convenience.

System domain rules. The domain description includes the declaration of the rewrite rules that represent the actions of the system. Some of the rules regard the actions of individual objects and are of the form:

$$<o{:}C|A|t> \xrightarrow{l}_{q} <o{:}C|A'|t>$$

i.e. the object o is able to perform an action with label l and cost q and the effect is reflected in its attributes. Note that such rules have no premise, i.e. they do not require any transition of sub-components. It is also usual to have unconditional rules involving more than one object (possibly with some nesting structure). For example, a basic rule of our scenario declares the ability of a ship to navigate (label `nav`) with a duration of 1 and a risk factor that depends on the weather conditions of both locations.

$$\begin{array}{l} < \texttt{o1} : \texttt{Location} \mid \texttt{weather} : \texttt{q1} \ , \ \texttt{a1} \mid \texttt{t1} > \ , \\ < \texttt{o2} : \texttt{Location} \mid \texttt{weather} : \texttt{q2} \ , \ \texttt{a2} \mid \texttt{t2} \\ \qquad < \texttt{o3} : \texttt{Ship} \mid \texttt{a3} \mid \texttt{t3} >> \\ \xrightarrow{\texttt{nav}}_{\langle 1, max\{q1,q2\}\rangle} < \texttt{o1} : \texttt{Location} \mid \texttt{weather} : \texttt{q1} \ , \ \texttt{a1} \mid \texttt{t1} \ , \\ \qquad\qquad\qquad\qquad < \texttt{o3} : \texttt{Ship} \mid \texttt{a3} \mid \texttt{t3} >> \ , \\ \qquad\qquad\qquad < \texttt{o2} : \texttt{Location} \mid \texttt{weather} : \texttt{q2} \ , \ \texttt{a2} \mid \texttt{t2} > \end{array}$$

Such individual actions are combined together with rules in the format discussed in § 4. Recall, that the actual choices of operations $\odot$ and $\oplus$ to combine action labels and quantitative information are crucial for the semantics of the rules. For instance, assume that our quantitative criteria includes action durations (which is the case of our case study) modelled with a tropical semiring. Several options are possible. If we let $\odot$ be the choice operation of the semiring (i.e.

min) we model the fact that the fastest action is considered (in which case it is meaningful to replace t_2' with t_2 in the conclusion of the rule). If we let $\odot$ be the join operation of the lattice underlying the semiring (i.e. *max*) we model the fact that the system has to wait to the slowest component to evolve. If we let $\odot$ be the combination operation of the the semiring (i.e. addition) we model the fact that system components evolve sequentially. Similar choices are possible for other quantitative dimensions. It is up to the designer to choose which one is more suitable in each case. As for label synchronisation, the standard approaches are possible like Hoare (all agree on the same action label) or Milner (complementary actions are synchronised) synchronisation. In our case study we tend to use Hoare synchronisation. For instance, the rule to combine the navigation of ships is

$$\frac{t_1 \xrightarrow{\texttt{nav}}_{q_1} t_1' \qquad t_2 \xrightarrow{\texttt{nav}}_{q_2} t_2'}{t_1 \ , \ t_2 \xrightarrow{\texttt{nav}}_{q_1(max,max)q_2} t_1' \ , \ t_2'}$$

i.e. we let two sets of ships navigate together at the slowest pace and considering the worst risk factor.

System problem description. With a fixed domain description, several instances are possible. An instance can be just a term of sort `Conf` (see e.g. Example 1 in §4) denoting the initial configuration of the system, but might include instance-dependent rules as well. Usually, a problem description will include a characterisation of goal configurations. For instance, in our case study, we have specified a function `isGoal` that characterises goal configurations, namely those configurations where all ships arrive to Stromboli (to tackle the emergency due to an increase of the eruptive activity of the vulcan).

Planning activities. In order to solve planning problems the system description must be imported from one of the planning theories discussed in §4. Then we can use Maude's `search` command to perform the corresponding reachability analysis. For instance, to find a solution for the rescue problem we can execute the command

```
search [1] initialConfiguration =>* reachableConfiguration:Conf
    such that isGoal(reachableConfiguration:Conf)
```

to obtain a goal state and, subsequently, the `show path` command to obtain a solution, as a sequence of system transitions, each made of the actions need to rescue the inhabitants of Stromboli, i.e. we obtain a rescue plan.

Instead, if we want to find optimal rescue plans we have to consider the theory of configuration sets and use the command

```
search [1] initialConfiguration =>* reachableConfigurations:Set{Conf}
    such that hasGoal(reachableConfigurations)
```

in which case we might obtain an absolute optimal rescue plan (one that is fastest and with less risk) or a non-dominated rescue plan (either one that is faster but involves more risk, or one less risky but slower).

6 Related work

We offer a brief discussion with related approaches that have influenced or inspired our work. A first source of inspiration is our previous work on *Architectural Design Rewriting* (ADR) [4] an approach that conciliates algebraic and graph-based techniques to offer a flexible model for software systems (e.g. software architectures) and their dynamics (e.g. architectural reconfiguration). Roughly, ADR models are rewrite theories interpreted over a particular class of hierarchical graphs. Another fundamental source of inspiration is the approach of [3], which provides a rewriting logic semantics to the Meta-Object Facility (MOF) and proposes the use of rewrite rules as a declarative description of model transformations. In a way, the present work conciliates both approaches. First, by enriching the formal model of [3] with explicit hierarchical features: in [3] compositions are modelled with references, so both models are somewhat homomorphic, but our explicit representation facilitates definitions (e.g. rules or predicates) by structural induction. Second, by devising a methodology inspired by our experience in modelling and analysing with ADR.

Similar approaches can be found in the field of quantitative process algebras (e.g. with applications to software architectures [1]), rewriting logic based quantitative specifications (e.g. timed [15] or probabilistic systems [11]) or quantitative model checking (see e.g. [10]). As far as we know, the focus has been on time and probabilistic/stochastic aspects in the tradition of Markovian models. It is possible to model some of such aspects with semiring as well but in a more approximated fashion. On the other hand, semirings offer various advantages: they are compositional, not limited to two aspects and enjoy algebraic properties that are inherent to many graph exploration algorithms, starting from the well known Floyd-Warshall's algorithm to solve the all-pairs shortest path problem. A particular variant of semirings has been implemented in Maude [9]. However, the variant of semiring used is slightly different from the one we need.

Our approach is also related to AI planning and in particular action planning. Due to space constraints we cannot offer a detailed overview of such a vast research field. However, we mention some prominent approaches. A relevant planning community centers around the Planning Domain Definition Language[9] (PDDL), a meta-language to be used as common problem domain and problem description language for various planning tools. The main difference with our to describe systems with inherently hierarchical aspects and does not allow to specify flexible (e.g. conditional) rules. However, many efforts have been invested towards expressiveness and performance. Interesting branches we are currently investigating are Temporal Planning [6] which copes with planning problems with durative, concurrent actions, and Hierarchical Task Planning [18] where the execution of a (hierarchical) task might require the execution of a plan of (sub) tasks.

[9] http://ipc.icaps-conference.org/

7 Conclusion

We have presented an approach for the description and analysis of model- and rule-based specifications with hierarchical structure. Our approach provides several benefits. First, it is built over the solid foundation of algebraic approaches like rewriting logic, structural operational semantics and semirings. Second, all the mathematical machinery is presented in the unifying, tool-supported framework of rewriting logic. Third, the approach fits perfectly with MOF-based technology as the MOF structure is somewhat homomorphic with our formalism. As a matter of fact our approach can be understood as a no-harm enhancement of the algebraic approach to MOF of [3]: one should be able to pass from a composition-as-relation representation to a composition-as-containment representation in a bijective manner, to use structural induction there where convenient. Fourth, the approach imposes a design discipline based on the hierarchical structure of composition associations, which contributes to the scalability of model- and rule-based approaches. Indeed, designers can benefit from the layered view introduced by the hierarchical structure and structured rewrite rules can lead to more efficient analysis activities.

In this regard, we are currently investigating performance comparisons between flat and structured approaches to model manipulations. In particular, preliminary results comparing classical examples of model transformations are very promising. Other current efforts regard the automatisation and tool-support for system descriptions. For instance, it should be possible to automatically derive the equational part of the theory from an UML class diagram, along the lines of the technique used in [3], e.g. deriving sorts from classes, subsorting from inheritance relations, and nesting from composition relations. In such way we could benefit from high-level front-ends.

We are also investigating more elaborated analysis problems. For instance, Maude provides a strategy language that we could use to restrict the set of acceptable plans we are interested in (e.g. by forbidding certain actions), and an LTL model checker that could be used for characterising plans with Linear-time Temporal Logic along the lines of *planning as model checking* approaches [8].

In future work we plan to conduct a comprehensive experimental evaluation to evaluate the applicability and scalability of our approach, considering automatically generated tests, industrial case studies, other application domains (e.g. architectural reconfiguration, refactoring) and comparison with state-of-the-art tools.

Acknowledgements. We are grateful to the organisers of the Dagstuhl Seminar on Graph Search Engineering (http://www.dagstuhl.de/09491) for their kind invitation to present some preliminary ideas, to Michele Sinisi and Raffaele Vertucci for the detailed, informal description of the case study, to the EU project SENSORIA and the the Italian project TOCAI.IT for their support, and to Priyan Chandrapala for his prototypical hierarchical editor of EMF models.

References

1. A. Aldini, M. Bernardo, and F. Corradini. *A process algebraic approach to software architecture design*. Springer, 2010.
2. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
3. A. Boronat and J. Meseguer. An algebraic semantics for MOF. In *Proceedings of the International Conference on Fundamental Aspects of Software Engineering (FASE'08)*, volume 4961 of *Lecture Notes in TCS*, pages 377–391. Springer, 2008.
4. R. Bruni, A. Lluch Lafuente, U. Montanari, and E. Tuosto. Style based architectural reconfigurations. *EATCS*, 94:161–180, 2008.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. *All About Maude*, volume 4350 of *LNCS*. Springer, 2007.
6. A. Coles, M. Fox, K. Halsey, D. Long, and A. Smith. Managing concurrency in temporal planning using planner-scheduler interaction. *Journal on Artificial Intelligence*, 173(1):1–44, 2009.
7. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, March 2006.
8. F. Giunchiglia and P. Traverso. Planning as model checking. In S. Biundo and M. Fox, editors, *Procceedings of the 5th European Conference on Planning (ECP'99)*, volume 1809 of *LNCS*, pages 1–20. Springer, 2000.
9. M. Hölzl, M. Meier, and M. Wirsing. Which soft constraints do you prefer? In *Proceedings of the 7th International Workshop on Rewriting Logic and its Applications (WRLA'08)*, volume 238(3) of *ENTCS*, pages 189–205. Elsevier, 2008.
10. J.-P. Katoen. Advances in probabilistic model checking. In G. Barthe and M. V. Hermenegildo, editors, *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2010)*, volume 5944 of *LNCS*. Springer, 2010.
11. N. Kumar, K. Sen, J. Meseguer, and G. Agha. A rewriting based model for probabilistic distributed object systems. In E. N. et al., editor, *International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'03)*, volume 2884 of *LNCS*, pages 32–46. Springer, 2003.
12. I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proceedings of the 3rd Joint Workshops on Foundations of Global Ubiquitous Computing (FGUC'04)*, volume 138(1) of *ENTCS*, pages 43–60. Elsevier, 2005.
13. A. Lluch Lafuente and U. Montanari. Quantitative mu-calculus and CTL defined over constraint semirings. *TCS*, 346(1):135–160, 2005.
14. J. Meseguer. Conditional rewriting logic as a united model of concurrency. *TCS*, 96(1):73–155, 1992.
15. P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. *TCS*, 285(2):359–405, 2002.
16. G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.
17. G. Rote. A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion). *Journal on Computing*, 34(3), 1985.
18. S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.
19. E. Seidewitz. What models mean. *IEEE Journal on Software*, 20(5):26–32, 2003.
20. A. Verdejo and N. Martí-Oliet. Executable structural operational semantics in Maude. *Journal of Logic and Algebraic Programming*, 67(1-2):226–293, 2006.