European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

# Dynamic Reconfiguration of GCM components

*A.Basso, A. Bolotov, V. Getov*
{A.Basso,A.Bolotov,A.Getov}@wmin.ac.uk
*University of Westminster, UK*

*L. Henrio*
{Ludovic.Henrio}@sophia.inria.fr
*INRIA, Sophia Antipolis, France*

CoreGRID Technical Report
Number TR-0173
September 19, 2008

Institute on Programming Model

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

# Dynamic Reconfiguration of GCM components

A.Basso, A. Bolotov, V. Getov
{A.Basso,A.Bolotov,A.Getov}@wmin.ac.uk
University of Westminster, UK

L. Henrio
{Ludovic.Henrio}@sophia.inria.fr
INRIA, Sophia Antipolis, France

*CoreGRID TR-0173*

September 19, 2008

**Abstract**

We detail in this report past research and current/future developments in formal specification of Grid component systems by temporal logic and consequent resolution technique, for an automated dynamic reconfiguration of components. It is analysed the specification procedure of GCM (Grid Component Model) components and infrastructure in respect to their state behaviour, and the verification process in a dynamic and reconfigurable distributed system. Furthermore it is demonstrated how an automata based method is used to achieve the specification, as well as how the enrichment of the temporal specification language of Computation Tree Logic CTL with the ability to capture norms, allows to formally define the concept of reconfiguration.

## 1 Introduction

There are two approaches to building long-lived and flexible Grid systems: exhaustive and generic. The former approach provides rich systems satisfying every service request from applications but consequently its implementation suffers from very high complexity. In the latter approach, we represent only the basic set of services (minimal and essential) and thus overcome the complexity of the exhaustive approach. However, to achieve the full functionality of the system, we must make this lightweight core platform reconfigurable and expandable. One of the possible solutions here is to identify and describe the basic set of features of the component model and to consider any other functions as pluggable components [31] which can be brought on-line whenever necessary [24].

In building generic Grid systems, we can take advantage of a basic representation of the system and build the required component parts on top as required. To take full advantage of this method, we have to ensure an easy and secure way to expand and reconfigure the platform. The overall complexity of this type of system is directly proportional to the type and amount of services implemented. Component models such as Fractal and the more specific Grid Component Model (GCM) allow for modular design of applications which can be easily reused and combined, ensuring greater reliability. This is important in distributed systems where asynchronous components must be taken into consideration, especially when the need for a dynamic reconfiguration is present. The GCM specification defines the basic (non-functional) controls to be implemented, and a number of constraints on the interplay between functional and non-functional operations. In these models, components interact together by being bound through interfaces.

Establishing the theoretical foundations of the generic processes involved in designing and functioning of such Grid systems is highly important. A significant part of this research lies in the area of formal specification of a component model and its infrastructure in relation to the dynamic state changes of such model, and its verification.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

For the specification of this behaviour we can use a rich temporal framework [21] with subsequent application of a verification technique. Given a formal specification of a distributed system there are two major approaches to formal verification of this specification: algorithmic and deductive [25]. While the algorithmic approach is fully automated, as in the case of model checking, its application, in general, is restricted to finite state systems. On the other hand, methods of the second, deductive approach, can handle arbitrary systems providing uniform proofs. To the best of our knowledge, the only technique currently used in the verification of distributed hierarchical components is model checking.

Note that a model checking technique, which verifies the properties of the components against the specification, has already been tested in various circumstances, one of which is described in [1]. While model checking [16] is a powerful and well established technique it has a particular drawback in its explorative nature, and makes considering the environment (especially during dynamic reconfiguration) impractical. Deductive methods on the other hand, can be applied on a large infrastructure-bound system and furthermore, can be integrated into reconfiguration scenarios. The technique we explored cannot replace model checking, as it is not intended for specification of behaviour of single components or static composition of components, on the other hand, it can complement model checking by assuming this part of the verification is already been done, and focusing on the dynamic state changes of a running Grid system and its current infrastructure.

In our approach the components are modelled in a specific branching-time temporal logic, or $\mathrm{SNF}_{\mathrm{CTL}}$ (Simplified Normal Form for Computation Tree Logic) [13], and then the temporal resolution is applied as a resolution tool. $\mathrm{SNF}_{\mathrm{CTL}}$ initially developed for CTL has been shown to be able to express simple fairness constraints and their Boolean combinations [7, 8]. Furthermore, a clausal resolution over the set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses has been defined [6, 7] and recently the search strategies for this method were presented in [9]. As a specific deductive method we have applied a temporal resolution method to a simple component model [3]. In [2] we proposed a framework for the specification of a reconfiguration process by deontic extension of the underlying language. However, the complexity of this approach turned out to be very high and one of the possible ways of its reduction would be to incorporate the developments in line with [20].

# 2 Background

## 2.1 Component Model

Fractal is a modular and extensible component model. The Grid Component model (GCM) is an extention of Fractal built to accommodate requirements in distributed systems, in particular the ones developed by the CoreGRID project. The GCM specification defines a set of notions characterising this model, an API (Application Program Interface), and an ADL (Architecture Description Language).

Components are characterised by their *content* and the *membrane*. The content of a component can be hidden (in which case it is simply a black box) or it can be constituted by a system of some other components (sub-components). In the former case we would call a component *primitive* while the latter case represents a *composite* component. The membrane, or controller, controls the component. *Controllers* address non-functional aspects of the component.

GCM is a multi-level specifications. Depending on their conformance level, GCM components can feature introspection and/or configuration. The *control* interfaces are used in the GCM model to allow configuration (reconfiguration), and are defined as *non functional*. On the other hand, the functional interfaces of a component are associated with its functionality. A *functional* interface can provide the required functionality and we call it the *server* interface. Alternatively, a *client* interface requires some other functionality.

Component interfaces are linked together by *bindings*. For this research, we will only consider primitive bindings that are simple bindings transmitting invocations from the client interface to the connected server interface.

There are few controllers that have been already defined in GCM (but others may be user-defined depending on the needs of the model); the most important are:

- The *attribute controller* is used to configure a property within a component, when there is no need to take into consideration bindings of interfaces.
- The *binding controller* is used when the attribute controller is not applicable and actual binding/unbinding of interfaces is required.

- The *content controller* can be used to retrieve the representation of the *sub components* and add or remove them accordingly; if a sub component is *shared* by one or more other components, the scenario must be defined so that also these other components are taken into consideration.
- The *life cycle controller* allows to start and stop a component, it is used for dynamic reconfiguration so that all other controls can be applied safely to the component while the component is not in execution.

## 2.2 Architecture

In the architecture developed, the formal specification is achieved by the analysis of three main parts: the primitive components, the composition of the former into composite components through the Architecture Description Language (ADL) file and the infrastructure (see Figure 1). The first two are combined to deduce the state behaviour of the component system - a high-level behaviour distinct from the one of a single component, which we assume to be already formally verified through other techniques being currently researched; the specification is partially given as an input by the user in the case of primitive components, and partially automatically extrapolated using different sources, such as source code and the ADL file. The infrastructure is specified mainly according to the user's need, and following well defined and accepted constrains such as those for safety, fairness, etc. [27] and in relation to the resources required and services provided. The formal specification derived through this process is a fusion of deontic and computation tree temporal logic, extended from the previous developments in [3], which is a suitable input format for our deductive reasoning tool. The properties to be specified and verified by this techniques are the ones which are not possible to be considered when a system is specified in a static way, this includes but is not limited to: presence of resources and services, availability of distributed components, etc.

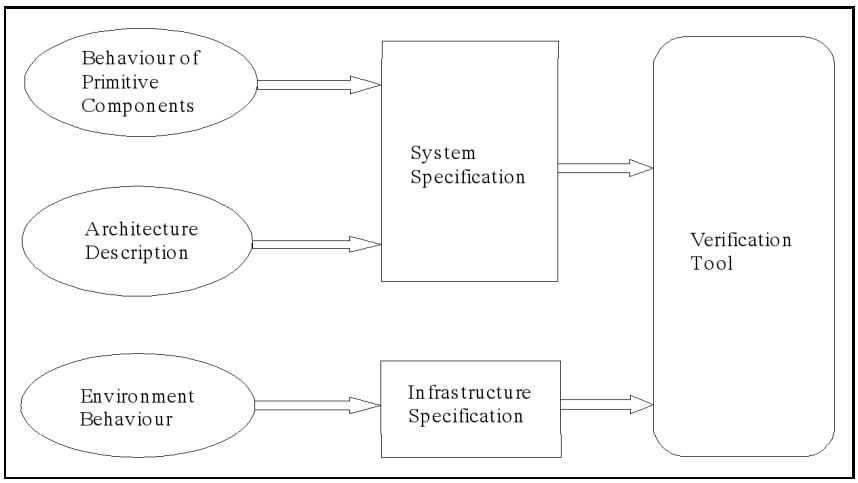


Figure 1: Architecture

**State behaviour specification - an example** In the classical approach to component behaviour specification, the term "behaviour" is referred to the inner component's functionality - if the component is supposed to calculate the factorial, is it doing it correctly? When we consider the state behaviour of a component instead, we are taking into consideration a different matter; we are looking for those requirements that will make the component "behave" correctly with the rest of the system. When thinking about a small example, this can be trivial - the parser can check if all the libraries required by the component are present to calculate the factorial - but what happens when we talk about a distributed system, where changes might be needed to be done at runtime? What if we require to replace a component but the component we want to replace depends on a resource which is not available? We have taken into consideration these types of situations while developing a specification procedure. We take into consideration the life cycle of a component and define its states in a formal way so that they can be used in the system specification. We consider past developments within the GCM and other state aware Grid systems [30] to define a set of states to be generated and could be monitored by specific software [17]. This lifecycle is restricted, in fact it only models the deployment state of the system (and consequentially its states transitions throughout the life of the system), not its

operational characteristics. For example, once a component is in running state, it is considered to be available. On the other hand, the service may fail for other unforeseen circumstances (hence the need for a component monitoring system during runtime which will report a need for changes into the state behaviour specification). We can represent these lifecycle states as in Figure 2.

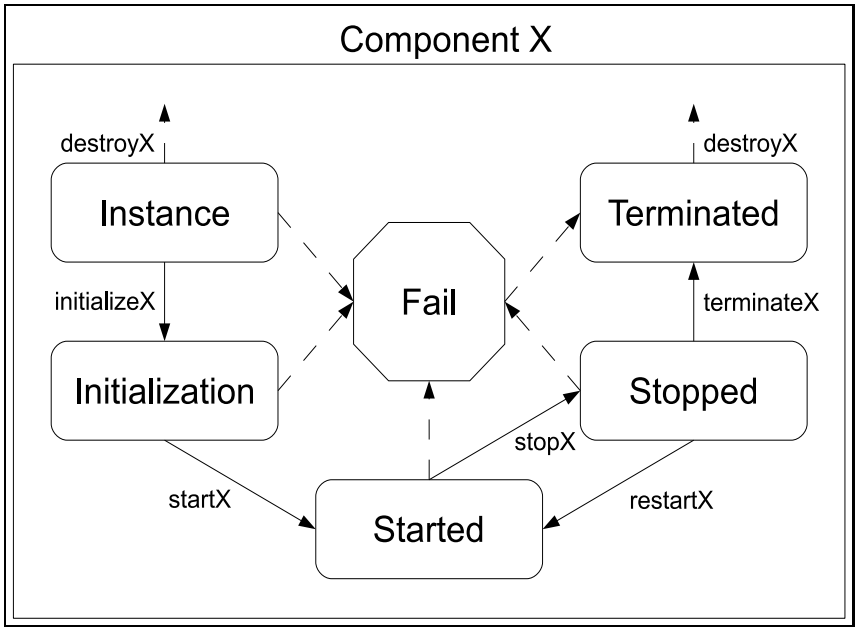


Figure 2: Component's Lifecycle States

In the GCM specification, the life-cycle controller is extended allowing to separate partially the life-cycle states of the controller and of the content. Generally, when a component created, it is firstly instantiated (deployment stage) and any operations (such as dependencies checks) are processed. In this state is assumed that the resource monitoring process will detect the resources required by the component and their presence. At this point the states changes initialise and destroy could be reached. The next state (assuming the component monitoring process has not detected a failure state) is the started state. This state is indicative for the services required/provided by the component/resources that are being deployed and available for use. This state does not refer to the internal operations of the component, but only the resources being used. Furthermore it should be clear that if a component would be reported at any state to have reached the failure state, the component itself may still be available to the system (although its required resources would become available), on the other hand the services provided by the component would be terminated. This applies to the terminated state as well. Due to the hierarchical composition of components, we can plot recursively resources required and services provided in the behavioural states system. When a component is functionally stopped (which corresponds to the stopped state of the Fractal specification), invocation on controller interfaces are enabled and the content of the component can be reconfigured. After the reconfiguration has taken place, the Fractal specification allows for the component to be restarted by using the start call on the component again.

**Automata Based Model** In building the specification protocol, we plan to follow well known automata constructions. We take a simple finite state automaton on finite strings, for the components specification, and a more complex finite state automata on infinite trees to define the environment. The automata at component level are used for the creation of labels defining the various states in which the considered component is, and are then fed upon request on to the various states of the automata at environment level, as shown in Figure 3.

In the construction of this tree automaton, every state is labelled according to state of components (passed over from the component level automaton) and resources. In this case the transition function is not only related to the state transition of components, but is also tightly bound to the deontic logic accessibility relation. Here we expect that we would be able to specify the automaton in the normal form for CTL, $\mathrm{SNF}_{\mathrm{CTL}}$, developed in [13]. Although we do not have a rigorous proof of this, we can anticipate that the situation here would be similar to the one in the linear-time case. Namely, in [12], it was shown that a Buchi word automaton can be represented in terms of $\mathrm{SNF}_{\mathrm{PLTL}}$, a normal

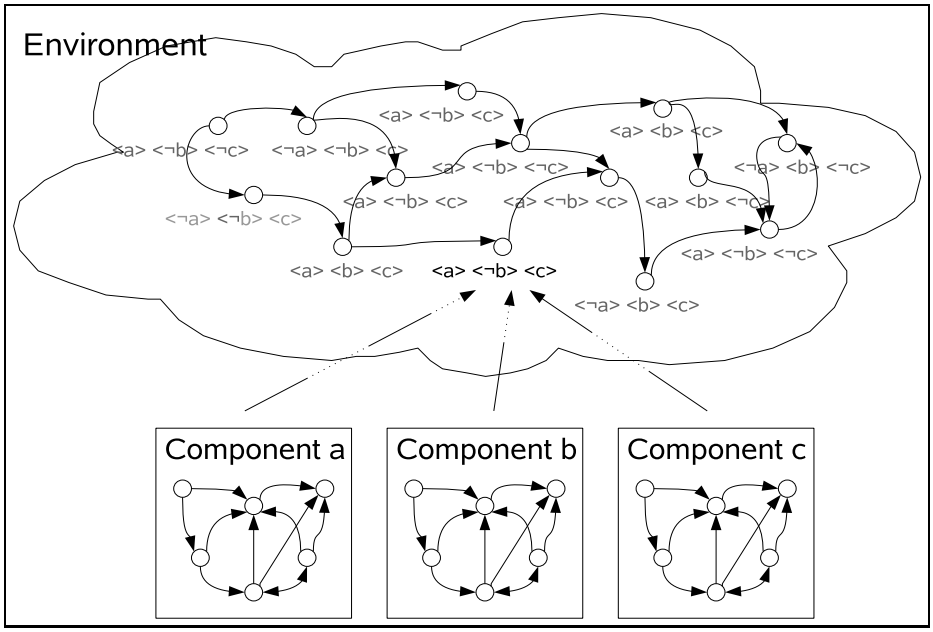


Figure 3: Automata Based Model

form for PLTL. Similarly, we can represent a Buchi tree automaton in terms of $\mathrm{SNF}_{\mathrm{CTL}}$. More precisely, we use $\mathrm{SNF}_{\mathrm{CTL}}$to specify the tree automaton and then extend this specification to a deontic temporal specification [2] and apply a resolution based verification technique as a verification procedure. As detailed in section 2.5, we assume that the models of CTL type logic are infinite trees, i.e. every path through such tree is an infinite path, allowing us to have these closure properties (suffix, limit and fusion closures) that make these models so called R-generable structures [21].

**System/Infrastructure Behaviour.** To extrapolate the state behaviour, we need information on the architecture and hierarchical components structure, its dynamic flow on information, the possible requirements and external requests etc. It is possible to extrapolate interface and bindings information from the XML based ADL file. This gives us an idea of the flow of the system; the user might need to refine this process, for example to keep significant parameters only or add relevant others. On the other hand, other component's requirements must be grabbed directly from the component's definitions. Since one of the GridComp IDE's functionality will include a GCM parser to build component models, we can reuse some of the data it provides to blueprint these requirements, leaving the task to the programmer to fill in the gaps. The infrastructure can represent a general purpose environment based on some common grounds, or a specific one, defined by the programmer. In the former case, infrastructure must, of course, leave room for further expansion and adaptation depending on the programmer's need.

**Deontic extension of specification.** We develop a specification language based on the fusion of Computation Tree Logic (CTL) and deontic logic to represent the properties of the behaviour protocol. The requirements of the protocol are understood as norms and specified in terms of deontic modalities "obligations" and "permission". The introduction of this deontic dimension not only increases the expressiveness of the system capturing the normative nature of the specification but also allows us to approach the reconfiguration problem in a novel way.

**Complexity.** We have mentioned that while our initial papers on the resolution based verification of the component model model specifications [3] opened a theoretical prospect of these developments, the complexity of the resolution based verification has raised some concerns with the feasibility of applying this method to a full scale component model. Therefore, there has been a need for complexity reduction. Unlike model checking, where the complexity lies in the model generation, deductive reasoning 'suffers' in the verification process. One of the ways to overcome the problem would be to develop a tractable sub-classes of propositional linear temporal logic, based on the use of XOR fragments of the logic following the linear time resolution framework in [20]. It should be also considered that the initial specification would not be as complex as in the most common uses of model checking, since the component's behaviour to be specified is only bound to the dynamic states: we do not consider the component's behaviour in the

classical sense (the functionality).

## 2.3 Reconfiguration

The reconfiguration aspect is an essential one in this research and we argue that our approach brings some important novelty comparing with other similar formal approaches to specification and verification. We need therefore to introduce this notion and give some definitions and descriptions on how we tackle the problem.

We refer to reconfiguration as to the process through which a system halts operations under its current source specification and begins operations under a different target specification [29]. Due to the underlying structure of generic Grid systems, and the particular nature of the reconfiguration process being developed, we consider the dynamic reconfiguration process as an unforeseen action at development time (known as ad-hoc reconfiguration [4]), therefore we will not be considering programmed reconfiguration. An insightful example could be the replacement of a software component by the user, or an automated healing process activated by the system itself. In the case that the system is not yet deployed, the verification of the overall system behaviour (inconsistency check) can be triggered manually at any step of the development process; the verification tool (VT) might simply detect inconsistencies and trigger the healing process or complete the verification and return to the user. When the system is deployed, the verification tool will run continuously and the system will report back the current states for model mapping; if a reconfiguration procedure is requested or inconsistency detected, the healing process is triggered. The inner behaviour of single components and their static composition is assumed to be pre-specified and verified using other techniques, such as model checking. We do not claim by any mean to be able to dynamically run a verification on the behaviour of each and every single component; we aim at a higher level of behaviour, the one of the overall Grid system, against the Grid infrastructure.

In both static and dynamic reconfiguration, there are three possible responses to a request for reconfiguration:

1. *Full rejection.* The VT rejects the reconfiguration request until the job is finished or there is a suitable change in the infrastructure. In this case, the VT might return to the user a suggestion on how the reconfiguration could be acceptable by suggesting a set of states which are accessible from the current state, in other words, the VT gives an 'if' condition which should be available to the system to accept the reconfiguration request.

2. *Full acceptance.* The VT accepts the request for reconfiguration without any constrains. This process assumes that the environment will not change in a affecting way during the required period for reconfiguration.

3. *Partial acceptance.* The VT accepts the request for reconfiguration but gives specific state-time constrains. In this case, need for automated reconfiguration is essential; the VT may also suggest a possible set of states which would allow for a broader acceptance (much like in the first case).

4. *Partial rejection.* Similarly to partial acceptance, the VT rejects the request for reconfiguration with some time constraints.

All of the above responses to requests for reconfiguration are intended for automated reconfiguration only - this is the only way to assure that the process takes place at acceptable state-time and infrastructure compliance time.

The dynamic reconfiguration process works in a circular way [Figure 4] and it is divided into two majors steps (detailed below) before the model update can take place. The approach here is to specify general invariants for the infrastructure and to accept any change to the system, as long as these invariants hold. We assume that the infrastructure has some pre-defined *set of norms* which define the constraints for the system, in order to ensure system safety, mission success, or other crucial system properties which are critical especially in distributed systems.

**Model update request.** A model update request can be triggered by a user's intention to re-configure the system, or by an inconsistency detection from the Verification Tool. It is reflected in the model as a change to the behaviour specification and it is constrained by the infrastructure restrictions. For example, the user might want to upgrade a component, but these changes must conform to the limitations set for such component. If the changes themselves are safe for the system, the Verification Tool passes to the next step.

**Model mapping.** For the verification process to understand its current state in the temporal tree, there is a need for a constant 'model mapping'; in other words, a process running in the background needs to be present in order to map the structure of the system into a model tree. This can be easily implemented alongside with a current monitoring system which will keep track of this mapping indicating which parts of the system are currently in which states in the model tree [5]. This process is essential to ensure that no 'past' states are misused by the VT during the healing process.

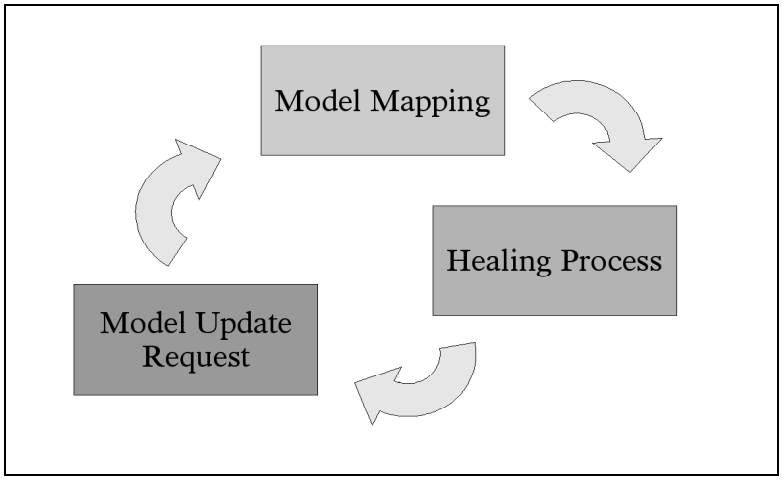


Figure 4: reconfiguration Cycle

## 2.4 Model Update (Healing process)

Model update is the core of the reconfiguration process. We refer to model update as to the process by which the state model, generated from the specification, is changed or partially replaced in order to respond to the update request. Unlike model revision, in which the description is simply corrected but the overall system remains unchanged, using model update assumes fundamental changes to the system by adding, deleting and replacing states in the model behaviour [23]. In this process, different types of changes are dealt with in a similar fashion, independently from the origin of the update (external user input or self healing process). To map the states set within where the model update can be applied, we apply the rules that formally define reconfiguration in section §2.3; assuming that specified norms and invariants do not contradict the new specification for the selected states set, the new set of states (and therefore the updated specification) is taken as the new model to be used and the updated component can be replaced by the software through standard unbinding etc.

**Healing process.** This process is triggered when the new set of updated states does not conform with the previous or following states. In other words, this process is used to search for a feasible modification to the rejected set of states, within the state model used. Formally, we search the tree model for a set of states which conform with the norms and invariants, and is applicable for this set of states. Candidate states for such an update in relation for some state $s_i$, do not have to be in an 'achievable' (from the tree order point of view) future of $s_i$, i.e. do not have to belong to a subtree with the root $s_i$, but only have to be 'accessible' from the current state according to the norms set ('deontically' accessible) by the infrastructure. The candidate set of states (or a more readable parsed version) is reported to the user/developer as a possible solution to the inconsistency detected. (healing is also triggered if there was no supplied update as in the case of inconsistency detection). This will open a prospect of re-configuring a model by making this 'deontically' accessible state also accessible in terms of the tree order. We are planning to incorporate here the ideas of a model update developed, in particular, in [18].

## 2.5 Temporal Deontic Specification

As the deontic extension of CTL we consider the logic which is described in [26]. We assume that the specification of a component model now is either written directly in this new framework of TDS or is initially given in the deontic extension of the logic ECTL$^+$ called ECTL$_D^+$ and then is converted into the TDS. Since the structure of TDS is similar to the SNF$_{\mathrm{CTL}}$ we are able to subsequently apply the resolution based verification technique which must be also extended to cope with the normative dimension.

### 2.5.1 The logic ECTL$_D^+$

In the language of ECTL$_D^+$, where formulae are built from the set, $Prop$, of atomic propositions $p, q, r, \ldots, p_1, q_1, r_1, \ldots, p_n, q_n, r_n, \ldots$, we use the following symbols: classical operators: $\neg, \wedge, \Rightarrow, \vee$; temporal operators: $\Box$ – 'always in the

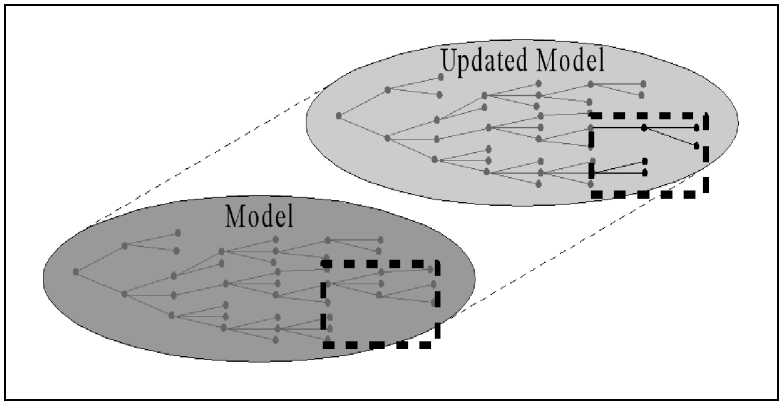


Figure 5: Model Update

future'; $\Diamond$ – 'at sometime in the future'; $\bigcirc$ – 'at the next moment in time'; $\mathcal{U}$ – 'until'; $\mathcal{W}$ – 'unless'; and path quantifiers: **A** – 'for any future path; **E** – 'for some future path.

For the deontic part we assume a set $Ag = \{a, b, c \ldots\}$ of agents (processes), which we associate with deontic modalities $\mathcal{O}_a(\varphi)$ read as '$\varphi$ is obligatory for an agent $a$' and $\mathcal{P}_a(\varphi)$ read as '$\varphi$ is permitted for an agent $a$'.

In the syntax of ECTL$_D^+$ we distinguish *state* ($S$) and *path* ($P$) formulae, such that $S$ are well formed formulae. These classes of formulae are inductively defined below (where $C$ is a formula of classical propositional logic)

$S ::= C|S \wedge S|S \vee S|S \Rightarrow S|\neg S|\mathbf{A}P|\mathbf{E}P|\mathcal{P}_a S|\mathcal{O}_a S$
$P ::= P \wedge P|P \vee P|P \Rightarrow P|\neg P|\,\Box S|\Diamond S|\bigcirc S|S\,\mathcal{U}\,S|S\,\mathcal{W}\,S|\,\Box\Diamond S|\Diamond\,\Box S$
Where path formulae are those that are evaluated along paths, and state formulae are those that are evaluated at the states of a tree.

**Definition 1 (literal, modal literal)** *A* literal *is either $p$, or $\neg p$ where $p$ is a proposition. A* modal literal *is either $\mathcal{O}_i l$, $\neg\mathcal{O}_i l$, $\mathcal{P}_i l$, $\neg\mathcal{P}_i l$ where $l$ is a literal and $i \in Ag$.*

ECTL$_D^+$ **Semantics.** We first introduce the notation of tree structures, the underlying structures of time assumed for branching-time logic.

**Definition 2** *A* tree *is a pair $(S, R)$, where $S$ is a set of states and $R \subseteq S \times S$ is a relation between states of $S$ such that $s_0 \in S$ is a unique root node, i.e. there is no state $s_i \in S$ such that $R(s_i, s_0)$; for every $s_i \in S$ there exists $s_j \in S$ such that $R(s_i, s_j)$; for every $s_i,\ s_j,\ s_k \in S$, if $R(s_i, s_k)$ and $R(s_j, s_k)$ then $s_i = s_j$.*

A *path*, $\chi_{s_i} \in P$ is a sequence of states $s_i, s_{i+1}, s_{i+2} \ldots$ such that for all $j \geq i$, $(s_j, s_{j+1}) \in R$. Let $\chi$ be the set of all paths of $\mathcal{M}$. A path $\chi_{s_0} \in \chi$ starting at the initial state is called a *fullpath*. Let $X$ be a family of all fullpaths of $\mathcal{M}$. Given a path $\chi_{s_i}$ and a state $s_j \in \chi_{s_i},\ (i < j)$ we term a finite subsequence $[s_i, s_j] = s_i, s_{i+1}, \ldots, s_j$ of $\chi_{s_i}$ *a prefix* of a path $\chi_{s_i}$ and an infinite sub-sequence $s_j, s_{j+1}, s_{j+2} \ldots$ of $\chi_{s_i}$ *a suffix* of a path $\chi_{s_i}$ abbreviated $Suf(\chi_{s_i}, s_j)$.

Following [21], without loss of generality, we assume that underlying tree models are of at most countable branching.

**Definition 3 (Total countable $\omega$-tree)** *A countable $\omega$-tree, $\tau_\omega$, is a tree $(S, R)$ with the family of all fullpaths, $X$, which satisfies the following conditions: each fullpath is isomorphic to natural numbers; every state $s_m \in S$ has a countable number of successors; $X$ is R-generable* [21], *i.e. for every state $s_m \in S$, there exists $\chi_n \in X$ such that $s_m \in \chi_n$, and for every sequence $\chi_n = s_0, s_1, s_2 \ldots$ the following is true: $\chi_n \in X$ if, and only if, for every $m$ $(1 \leq m)$, $R(s_m, s_{m+1})$.*

Since in $\omega$ trees fullpaths are isomorphic to natural numbers, in the rest of the paper we will abbreviate the relation $R$ as $\leq$.

Next, for the interpretation of deontic operators, we introduce a binary agent accessibility relation.

**Definition 4 (Deontic Accessibility Relation)** *Given a total countable tree $\tau_\omega = (S, \leq)$, a binary agent accessibility relation $D_i \subseteq S \times S$, for each agent $i \in Ag$, satisfies the following properties: it is* serial *(for any $k \in S$, there exists $l \in S$ such that $D_i(k, l)$),* transitive *(for any $k, l, m \in S$, if $D_i(k, l)$ and $D_i(l, m)$ then $D_i(k, m)$), and* Euclidian *(for any $k, l, m \in S$, if $D_i(k, l)$ and $D_i(k, m)$ then $D_i(l, m)$).*

Let $(S, \leq)$ be a total countable $\omega$-tree with a root $s_0$ defined as in Def 3, $X$ be a set of all fullpaths, $L : S \times Prop \longrightarrow \{\mathbf{true}, \mathbf{false}\}$ be an interpretation function mapping atomic propositional symbols to truth values at each state, and every $R_i \subseteq S \times S$ $(i \in 1, \ldots, n)$ be an agent accessibility relation defined as in Def 4. Now a model structure for interpretation of ECTL$_D^+$ formulae is $\mathcal{M} = \langle S, \leq, s_0, X, L, D_1, \ldots, D_n \rangle$.

Reminding that since the underlying tree structures are $R$-generable, they are suffix, fusion and limit closed [21], in Figure 6 we define a relation '$\models$', which evaluates well-formed ECTL$_D^+$ formulae at a state $s_m$ in a model $\mathcal{M}$.

| | | | |
|---|---|---|---|
| $\langle \mathcal{M}, s_m \rangle$ | $\models p$ | iff | $p \in L(s_m)$, for $p \in Prop$ |
| $\langle \mathcal{M}, s_m \rangle$ | $\models \mathbf{A}B$ | iff | for each $\chi_{s_m}, \langle \mathcal{M}, \chi_{s_m} \rangle \models B$ |
| $\langle \mathcal{M}, s_m \rangle$ | $\models \mathbf{E}B$ | iff | there exists $\chi_{s_m}$ such that $\langle \mathcal{M}, \chi_{s_m} \rangle \models B$ |
| $\langle \mathcal{M}, \chi_{s_m} \rangle$ | $\models A$ | iff | $\langle \mathcal{M}, s_m \rangle \models A$, for state formula $A$ |
| $\langle \mathcal{M}, \chi_{s_m} \rangle$ | $\models \Box B$ | iff | for each $s_n \in \chi_{s_m}$, if $m \leq n$ then $\langle \mathcal{M}, Suf(\chi_{s_m}, s_n) \rangle \models B$ |
| $\langle \mathcal{M}, \chi_{s_m} \rangle$ | $\models \bigcirc B$ | iff | $\langle \mathcal{M}, Suf(\chi_{s_m}, s_{m+1}) \rangle \models B$ |
| $\langle \mathcal{M}, \chi_{s_m} \rangle$ | $\models A \,\mathcal{U}\, B$ | iff | there exists $s_n \in \chi_{s_m}$ such that $m \leq n$ and $\langle \mathcal{M}, Suf(\chi_{s_m}, s_n) \rangle \models B$ and for each $s_k \in \chi_{s_m}$, if $m \leq k < n$ then $\langle \mathcal{M}, Suf(\chi_{s_m}, s_k) \rangle \models A$ |
| $\langle \mathcal{M}, \chi_{s_m} \rangle$ | $\models A \,\mathcal{W}\, B$ | iff | $\langle \mathcal{M}, \chi_{s_m} \rangle \models \Box A$ or $\langle \mathcal{M}, \chi_{s_m} \rangle \models A \,\mathcal{U}\, B$ |
| $\langle \mathcal{M}, s_m \rangle$ | $\models \mathcal{O}_a B$ | iff | for each $s_n \in S$, if $D_a(m, n)$ then $\langle \mathcal{M}, s_n \rangle \models B$ |
| $\langle \mathcal{M}, s_m \rangle$ | $\models \mathcal{P}_a B$ | iff | there exists $s_n \in S$, such that $D_a(m, n)$ and $\langle \mathcal{M}, s_n \rangle \models B$ |

Figure 6: ECTL$_D^+$ semantics

**Definition 5 (Satisfiability)** *A well-formed* ECTL$_D^+$ *formula, $B$, is satisfiable if, and only if, there exists a model $\mathcal{M}$ such that $\langle \mathcal{M}, s_0 \rangle \models B$.*

**Definition 6 (Validity)** *A well-formed* ECTL$_D^+$ *formula, $B$, is valid if, and only if, it is satisfied in every possible model.*

### 2.5.2 Temporal Deontic Specification

In this section we introduce the core concept of Temporal Deontic Specification (TDS).

**Definition 7 (Temporal Deontic Specification - TDS)** *TDS is a tuple $\langle In, St, Ev, N, Lit \rangle$ where $In$ is the set of initial constraints, $St$ is the set of step constraints, $Ev$ is the set of eventuality constraints, $N$ is a set of normative expressions, and $Lit$ is the set of literal constraints, i.e. formulae that are globally true. The structure of these constraints called* clauses*, is defined below where each $\alpha_i$, $\beta_m$, $\gamma$ or $l_e$ is a literal,* **true** *or* **false***, $d_e$ is either a literal or a modal literal involving the $\mathcal{O}$ or $\mathcal{P}$ operators, $\langle \mathsf{ind} \rangle \in \mathsf{IND}$ is some index, and the clauses are supposed to be in the scope of the* $\mathbf{A}\Box$ *modality.*

$$\mathbf{start} \Rightarrow \bigvee_{i=1}^{k} \beta_i \quad \text{(In)}$$

$$\bigwedge_{i=1}^{k} \alpha_i \Rightarrow \mathbf{A}\bigcirc \left[\bigvee_{m=1}^{n} \beta_m\right] \quad \text{(St } \mathbf{A}\text{)}$$

$$\bigwedge_{i=1}^{k} \alpha_i \Rightarrow \mathbf{E}\bigcirc \left[\bigvee_{m=1}^{n} \beta_m\right]_{\langle \mathsf{ind} \rangle} \quad \text{(St } \mathbf{E}\text{)}$$

$$\bigwedge_{i=1}^{k} \alpha_i \Rightarrow \mathbf{A}\Diamond\gamma \quad \text{(Ev } \mathbf{A}\text{)}$$

$$\bigwedge_{i=1}^{k} \alpha_i \Rightarrow \mathbf{E}\Diamond\gamma_{\langle \mathsf{LC(ind)} \rangle} \quad \text{(Ev } \mathbf{E}\text{)}$$

$$\mathbf{true} \Rightarrow \bigvee_{e=1}^{n} d_e \quad \text{(D)}$$

$$\mathbf{true} \Rightarrow \bigvee_{e=1}^{n} l_e \quad \text{(Lit)}$$

In the rest of the paper, similar to our previous presentations, to simplify reading, we will omit writing the common for all TDS clauses prefix $\mathbf{A}\,\Box$.

### 2.5.3 Rules towards TDS

We suppose that the given specification is either directly written in terms of TDS or in the language of ECTL$^+_D$. In the latter case the formulae of the specification must be transformed into the desired form of TDS. Since ECTL$^+_D$ extends ECTL$^+$ by allowing deontic constraints and similarly since TDS is a deontic extension of SNF$_{\mathrm{CTL}}$, a normal form for the logic ECTL$^+$, [8] we simply enrich the transformation procedure of [8] by the corresponding rules dealing with the deontic operations. In fact, due to the fact that there is no interaction between temporal and deontic constraints, the only rule that is needed for the transformation of the formulae with deontic constraints is the renaming rule, which would work similar to [19], i.e. which would allow us to rename an embedded deontic subformula by a new proposition. Thus, to assist the reader we will only review those rules of the transformation procedure that are used in our example in section §2.5.4 and section §2.6.2.

**Renaming**

Given $P \Rightarrow Q(F)$ we derive $P \Rightarrow Q(F/x)$ and $x \Rightarrow F$, where $Q(F)$ is a formula with the designated subformula $F$ and $Q(F/x)$ means a result of replacing $F$ by a new proposition symbol $x$ in $Q$.

**Temporising**

Give a purely classical expression $A \Rightarrow B$ we transform it into $\mathbf{start} \Rightarrow \neg A \vee B$ and $\mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg A \vee B)$. In particular, we will apply the following case of this rule: from $\mathbf{true} \Rightarrow A \vee B$ derive $\mathbf{start} \Rightarrow \neg A \vee B$ and $\mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg A \vee B)$.

**Removal of $\mathbf{A}\,\mathcal{W}$**

Recall that the formula of the type $\mathbf{A}(A\,\mathcal{W}\,B)$ or $\mathbf{E}(A\,\mathcal{W}\,B)$ can appear as a result of the application of Temporal Resolution rules. Hence, we would have to transform a resolvent of this type into the desired form. This is first of all achieved by the application of the $\mathcal{W}$ removal rule. In particular, we would apply the $\mathbf{A}\,\mathcal{W}$ removal rule: given $P \Rightarrow \mathbf{A}(A\,\mathcal{W}\,B)$ derive $P \Rightarrow B \vee (A \wedge x)$ and $x \Rightarrow \mathbf{A}\bigcirc(B \vee (A \wedge x))$ [6].

### 2.5.4 TDS Example

Now, let us consider an example specification in which we essentially use a normative framework for reconfiguration, and where a model is requested to be updated.

Let $r$ and $s$ represent two components that can be bound to the system. Further let $q$ be a new composite component, a composition of $r$ and $s$. Next, let $r$ and $s$ be such that $r$ always requires its counterpart component, $s$ not to be active in any of the next states and $s$ requires $r$ not to be bound in some possible development of the system, i.e. at the successor state in the direction $\langle f \rangle$. Additionally, let us assume that the specification of the system requires that this new component, $q$, should not be bound at the next state. This is represented by the following formula of ECTL$^+_D$:

$$(\dagger) \qquad \mathbf{A}\,\Box(r \Rightarrow (\mathbf{A}\bigcirc s \wedge \mathcal{O}_i\neg q)) \wedge \mathbf{A}\,\Box(s \Rightarrow (\mathbf{E}\bigcirc r \wedge \mathcal{O}_i\neg q))$$

Finally, let the system receive a request for the permission to eventually bind $q$ whichever way it evolves:

$$(\ddagger) \qquad \mathbf{start} \Rightarrow \mathbf{A}\diamondsuit\mathcal{P}_i q$$

Our specification technique require to transform formulae $(\dagger)$ and $(\ddagger)$ to the structure required by TDS. This translation, as mentioned above, when it concerns with the temporal part, is described in our previous work [3, 8] and if it involves deontic constraints then we additionally use standard classical transformations towards normal forms and the renaming rule. To simplify the reading of the paper, we have already presented the rules involved into our examples below.

In the table below we summarise these conditions of the component system and their representations in the language of TDS (note that $w$ is a new (auxiliary) proposition introduced to achieve the required form of TDS clauses). Recall that each clause of TDS is in the scope of the $\mathbf{A}\,\Box$ and we will omit this common prefix for the TDS clauses in the rest of the paper to simplify the reading. Also, recall that each **E** step clause would have to be labelled by a specific label $f$ while every **E** sometime clause by some index $LC(ind)$ [6].

| Conditions of the System | Constraints of TDS |
|---|---|
| Dependency between counterpart components | $r \Rightarrow \mathbf{A}\bigcirc s$<br>$\mathbf{true} \Rightarrow \neg r \vee \mathcal{O}_i \neg q$<br>$s \Rightarrow \mathbf{E}\bigcirc r$<br>$\mathbf{true} \Rightarrow \neg s \vee \mathcal{O}_i \neg q$ |
| A request for the permission to eventually bind $q$ | $\mathbf{start} \Rightarrow x$<br>$x \Rightarrow \mathbf{A}\diamondsuit w$<br>$\mathbf{true} \Rightarrow \neg w \vee \mathcal{P}_i q$ |

## 2.6 Resolution based deduction verification of TDS

### 2.6.1 Resolution Rules

We first update the set of resolution rules developed for $\mathrm{SNF}_{\mathrm{CTL}}$ [13] by new resolution rules capturing the deontic constraints. However, due to the lack of space, here we present only those rules that will be involved into an example of the refutation. Recall that among the set of the TDS clauses are initial clauses, step clauses, eventuality formulae, and deontic clauses. In order to achieve refutation we apply three types of resolution rules: Step Resolution (classical resolution) (SRES), Temporal Resolution (TRES), and deontic resolution (DRES). SRES and TRES rules are described in [6].

Two step resolution rules that will be used in our example are given below.

$$\frac{\begin{array}{l}SRES1\\ \mathbf{start} \Rightarrow l \vee B\\ \mathbf{start} \Rightarrow l \vee C\end{array}}{\mathbf{start} \Rightarrow B \vee C} \qquad \frac{\begin{array}{l}SRES2\\ A \Rightarrow \mathbf{A}\bigcirc(l \vee D)\\ B \Rightarrow \mathbf{A}\bigcirc(\neg l \vee E)\end{array}}{A \wedge B \Rightarrow \mathbf{A}\bigcirc(D \vee E)}$$

When TDS clauses contain eventualities then the resolution procedure tackles the cases where such promises for the events to occur contradict some invariants, or loops [6]. We only note here that loops are formulae that constrain some proposition to be always true (on all or some paths) given some conditions hold. Finally, when two deontic clauses contain complimentary constraints, $O_i l$ and $P_i \neg l$ then we apply the new, deontic resolution rule, which, in fact, works similarly to the modal resolution rule in [19].

$$\frac{\begin{array}{l}DRES\\ \mathbf{true} \Rightarrow D \vee O_i l\\ \mathbf{true} \Rightarrow D' \vee P_i \neg l\end{array}}{\mathbf{true} \Rightarrow D \vee D'} \qquad \frac{\begin{array}{l}TRES\\ A \Rightarrow \mathbf{E}\,\Box \neg l_{\langle \mathsf{f} \rangle}\\ B \Rightarrow \mathbf{A}\diamondsuit l\end{array}}{B \Rightarrow \mathbf{A}(\neg A\, \mathcal{W}\, l)}$$

The resolvent of the TRES rule indicates that given the premises of the rule, i.e. a loop in $\neg l$ on all paths and to avoid a contradiction, we do not want the conditions, $A$, for the loop to occur unless $l$ has been satisfied (see [6] for more details and explanation why in this particular case we have rather unusual distribution of the path quantifiers in the premises and conclusion).

### 2.6.2 Resolution Example

Here we present a resolution based refutation for the set of clauses of TDS obtained for the component system analysed in the previous section.

| | $TDS$ | | |
|---|---|---|---|
| 1. | $r$ | $\Rightarrow$ | $\mathbf{A}\bigcirc s$ |
| 2. | $\mathbf{true}$ | $\Rightarrow$ | $\neg r \vee \mathcal{O}_i \neg q$ |
| 3. | $s$ | $\Rightarrow$ | $\mathbf{E}\bigcirc r_{\langle f \rangle}$ |
| 4. | $\mathbf{true}$ | $\Rightarrow$ | $\neg s \vee \mathcal{O}_i \neg q$ |
| 5. | $\mathbf{start}$ | $\Rightarrow$ | $x$ |
| 6. | $x$ | $\Rightarrow$ | $\mathbf{A}\diamondsuit w$ |
| 7. | $\mathbf{true}$ | $\Rightarrow$ | $\neg w \vee \mathcal{P}_i q$ |

| | $Proof$ | | | |
|---|---|---|---|---|
| 8. | $\mathbf{true}$ | $\Rightarrow$ | $\neg s \vee \neg w$ | $DRES\,4, 7$ |
| 9. | $r$ | $\Rightarrow$ | $\mathbf{A}\bigcirc \neg w$ | $SRES\,2,\ 1, 8$ |
| 10. | $s$ | $\Rightarrow$ | $\mathbf{A}\bigcirc \neg w$ | $from\ 8$ |
| 11. | $r \vee s$ | $\Rightarrow$ | $\mathbf{E}\bigcirc \Box \neg w_{\langle f \rangle}$ | $1, 3, 9, 10$ |
| 12. | $x$ | $\Rightarrow$ | $\neg(r \vee s)\,\mathcal{W}\,w$ | $TRES,\ 6, 11$ |
| 13. | $x$ | $\Rightarrow$ | $w \vee \neg(r \vee s)$ | $\mathcal{W}\ removal,\ 12$ |
| 14. | $x$ | $\Rightarrow$ | $w \vee \neg r$ | $classical, 13$ |
| 15. | $x$ | $\Rightarrow$ | $w \vee \neg s$ | $classical, 13$ |
| 16. | $\mathbf{start}$ | $\Rightarrow$ | $\neg x \vee w \vee \neg r$ | $temporising, 14$ |
| 17. | $\mathbf{start}$ | $\Rightarrow$ | $\neg x \vee w \vee \neg s$ | $temporising, 15$ |
| 18. | $\mathbf{start}$ | $\Rightarrow$ | $\neg s \vee \neg w$ | $temporising, 8$ |
| 19. | $\mathbf{start}$ | $\Rightarrow$ | $\neg s$ | $SRES 1,\ 5, 17, 18$ |

In this proof, step 8 is obtained by the application of deontic resolution to 4 and 7. Step 9 is the application of Step Resolution to 1 and 8 (recall that from $\mathbf{true} \Rightarrow \neg s \vee \neg w$ by temporising we can derive $\mathbf{start} \Rightarrow \neg s \vee \neg w$ and $\mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg s \vee \neg w)$, and we use the latter to resolve with 1). Thus, from $\mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg s \vee \neg w)$ we also have step 10.

Now, since we have an eventuality clause 6, $x \Rightarrow \mathbf{A}\diamondsuit w$, the resolution based verification technique searches for a loop in $\neg w$. The desired loop can be found combining together clauses 1, 3, 9 and 10 to give us: $r \vee s \Rightarrow \mathbf{E}\bigcirc \Box \neg w_{\langle f \rangle}$. This loop being resolved with the eventuality clause, produces the resolvent on step 12. Removing $\mathcal{W}$ from this resolvent, we deduce 13. Subsequent classical transformation and the temporising rule guide the deduction of steps 14-18. Finally, applying Step Resolution, we derive step 19.

In the next section we will show how these refutation is linked to the model update procedures.

## 2.7 Model Update Example

As we mentioned, we can describe how our system can identify normative invariants which should be preserved, we can also detect hidden invariants, i.e. those that are not explicitly given in the specification.

Analysing the proof in the previous section we know that $s$ should not be initially active. Note that our procedure has detected a loop (invariant) in $\neg w$ which is immediately obvious from the set of TDS clauses. Additionally, this loop, in conjunction with clauses 2 and 7 indicates a hidden 'deontic invariant' property, that $s$ fires the condition $\mathcal{O}_i \neg q$ and $w$ fires the condition $\mathcal{P}_i q$. Now, if we assume that $r$ is initially active, then we can continue the proof contained in the previous section and derive a contradiction as follows:

| | | | | |
|---|---|---|---|---|
| 20. | $\mathbf{start}$ | $\Rightarrow$ | $r$ | $assumption$ |
| 21. | $\mathbf{start}$ | $\Rightarrow$ | $w$ | $SRES\ 1,\ 1, 16, 20$ |
| 22. | $\mathbf{start}$ | $\Rightarrow$ | $\mathcal{P}_i q$ | $SRES 1,\ 7, 21$ |
| 23. | $\mathbf{start}$ | $\Rightarrow$ | $\mathcal{O}_i \neg q$ | $SRES 1,\ 2, 20$ |
| 24. | $\mathbf{start}$ | $\Rightarrow$ | $\mathbf{false}$ | $DRES,\ 22, 23$ |

Thus, a request to bring a composite component, $q$ to the system can only be satisfied if $r$ is not active. Otherwise, if $r$ is bound to the system, the request to bind $q$ should be rejected.

In our deontic language, we can set up some predefined accessibility (on the top of those that are defined for every model - such as transitivity, etc) which we call deontically accessible worlds During the reconfiguration we want to arrive at deontically accessible world to update our model, we can do this in two ways:

1. When such a world that corresponds to the reconfiguration specification can be found in the model and it is deontically accessible
2. When we cannot find such a world we want to update the model, this update should then satisfy both, the criteria of reconfiguration and this deontic accessibility

# 3 Conclusions and Future Works

There are some open problems and future works that need to be considered from this point on. In this section, we outline these aspects and give some ideas on the path that will be taken to solve these tasks.
So far, the following aspects have been tackled:

- Proposed a specification/verification framework [3].
- Analysed different deductive techniques as verification (resolution/natural deduction) tools.
- Extended propositional specification language by norms and proved the correctness of deontic temporal resolution (in relation to the axiomatics of temporal deontic logic). This method simulates earlier developed method for logic of rational agency by combining temporal and belief constraints.
- Investigated states of components and proposed relevant specification.
- Proposed an automata based approach for the whole model (which will need to be further analysed to enable this approach).
- Proposed a general scenario of using temporal deontic specification/verification for reconfiguration (which will need to be researched in detail, in particular for the model update).

The following are some issues and incomplete tasks which are left to solve:

1. Identify useful assumptions we can make in the specification language which currently does not allow for presentation of properties/relationships as first order language. The obstacle in applying only propositional language needs to be avoided; first order plus temporal framework is not only undecidable but also incomplete, and could be solved by thinking of fragments of first order temporal logic that are "nice" (i.e. decidable); it must be defined what exactly can be represented/specified using propositional language, which is the base for the analysis of an "average case".
2. Further analyse the aspect of automata based model through the use of states of components and automata. So far we have been working under the assumption that the automaton on the bottom layer simply goes through the component's states (i.e. the states of components are the states of the automaton), but the specification here may result to be more complex.
3. Another aspect to be considered in automata based representation, is how exactly the two automata will interact. We have not yet specified how the bottom layer automaton feeds the environmental one (i.e. how this passing of labels is carried on); further on this is to define exactly the tree automaton which would work on the environmental level. Unlike linear-time, where different type of automata on infinite words simulate each other, in the case of tree automata we can consider a few alternatives; depending on the type of tree automata there will be a need to prove that extended normal form can simulate such automaton.
4. Finally, the reconfiguration scenario should be explored in detail. It is not yet clear how we derive a model update, which was initially thought to derive a guide for the update from the results of the resolution verification.

# References


[1] T. Barros and L. Henrio and E. Madelaine. Verification of Distributed Hierarchical Components. In *Proceedings of the International Workshop on Formal Aspects of Component Software (FACS'05)*. Electronic Notes in Theoretical Computer Science (ENTCS). Macao, 2005, Oct.

[2] A. Basso and A. Bolotov. Towards GCM reconfiguration - extending specification by norms. To appear in: CoreGRID Springer Volume of the CoreGRID Workshop at Heraklion, June 2007.

[3] A. Basso, A. Bolotov, A. Basukoski, V. Getov, L. Henrio and M. Urbanski. Specification and Verification of Reconfiguration Protocols in Grid Component Systems. In *Proceedings of the 3rd IEEE Conference On Intelligent Systems IS-200*, 2006, IEEE.

[4] Batista, T., Joolia, A. and Coulson, G. Managing Dynamic Reconfiguration in Component-based Systems Proceedings of the European Workshop on Software Architectures, June, 2005, Pisa,Italy, Springer-Velag LNCS series, Vol 3527, pp 1-18.

[5] F. Baude and D. Caromel and F. Huet and L. Mestre and J. Vayssiere Interactive and Descriptor-Based Deployment of Object-Oriented Grid Applications HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 isbn 0-7695-1686-6, p93, IEEE Computer Society.

[6] A. Bolotov. Clausal resolution for extended computation tree logic ECTL. In *Proceedings of the Time-2003/International Conference on Temporal Logic 2003*, Cairns, July 2003. IEEE.

[7] A. Bolotov and A. Basukoski. Clausal resolution for extended computation tree logic ECTL. *Journal of Applied Logic*, in Press.

[8] A. Bolotov and A. Basukoski. A Clausal Resolution Method for Branching Time Logic ECTL+. Annals of Mathematics and Artificial Intelligence, Springer Verlag, in Press.

[9] A. Bolotov and A. Basukoski. Search Strategies for Resolution in CTL-Type Logics: Extension and Complexity. In *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning, (TIME 2005)* 195 - 197, IEEE Computer Society, 2005.

[10] A. Bolotov, A. Basukoski, O. Grigoriev, and V. Shangin. Natural deduction calculus for linear-time temporal logic. In *Joint European Conference on Artificial Intelligence (JELIA-2006)*, pages 56–68, 2006.

[11] A. Bolotov, V. Bocharov, A. Gorchakov, and V. Shangin. Automated first order natural deduction. In *Proceedings of IICAI*, pages 1292–1311, 2005.

[12] A. Bolotov, C. Dixon and M. Fisher. On the Relationship between Normal Form and W-automata (with M.Fisher and C.Dixon). Journal of Logic and Computation, Volume 12, Issue 4, August 2002, pp. 561-581, Oxford University Press.

[13] A. Bolotov and M. Fisher. A Clausal Resolution Method for CTL Branching Time Temporal Logic. *Journal of Experimental and Theoretical Artificial Intelligence.*, 11:77–93, 1999.

[14] A. Bolotov, O. Grigoriev, and V. Shangin. Automated natural deduction for propositional linear-time temporal logic. In *To be published in the Proceedings of the Time-2007, International Symposium on Temporal Representation and Reasoning*, June, 2007.

[15] A. Bolotov, O. Grigoriev, and V. Shangin. Natural deduction calculus for computation tree logic. In *IEEE John Vincent Atanasoff Symposium on Modern Computing*, pages 175–183, 2006.

[16] E. M. Clarke, A. Fehnker, S. Jha and H. Veith. Temporal Logic Model Checking., Handbook of Networked and Embedded Control Systems, 2005, pages 539-558.

[17] CoreGRID Programming Model Institute Basic Features of the Grid Component Model Deliverable D.PM.04, CoreGRID, March 2007.

[18] Y. Ding and Y. Zhang. CTL model update: Semantics, computations and implementation. In Proceedings of the 17th Europen Conference on Artificial Intelligence (ECAI 2006), pp 362-366. IOS Press 2006.

[19] C. Dixon, M. Fisher and A. Bolotov. Resolution in a Logic of Rational Agency. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000)*, Berlin, Germany.

[20] C. Dixon, M. Fisher, and B. Konev. Tractable Temporal Reasoning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 318-323, January 6-12th 2007, Hyderabad, India.

[21] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics.*, pages 996–1072. Elsevier, 1990.

[22] E. A. Emerson and A. P. Sistla. Deciding full branching time logic. In *STOC 1984, Proceedings of*, pages 14–24, 1984.

[23] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. PODS '92: Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium, p261-273, 1992, citeseer.ist.psu.edu/eiter92complexity.html.

[24] C. Goble, D. De Roure, N.R. Shadbolt, and A.A.A. Fernandes. Enhancing Services and Applications with Knowledge and Semantics. In *I. Foster and C. Kesselman, eds. The Grid 2: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann*, 2004.

[25] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic verifications. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP-98)*, Vol. 1443 of Lecture Notes in Computer Science, Springer:1-16,1998.

[26] A. Lomuscio and B. Wozna. A complete and decidable axiomatisation for deontic interpreted systems. In *DEON*, volume 4048 of *Lecture Notes in Computer Science*, pages 238–254. Springer, 2006.

[27] Z. Manna and A. Pnueli. Temporal Specification and Verification of Reactive Modules. Weizmann Institute of Science Technical Report, March 1992.

[28] A. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic.* PhD thesis, College of Science and Engineering, School of Informatics, University of Edinburgh, 1994.

[29] E. A. Strunk and J. C. Knight. Assured Reconfiguration of Embedded Real-Time Software. DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks, 2004, isbn 0-7695-2052-9, p367, IEEE Computer Society, Washington, DC, USA.

[30] J. Tatemura CDDLM Configuration Description Language Specification GFD-R-P.085, August 2006.

[31] J. Thiyagalingam, S. Isaiadis and V. Getov. Towards Building a Generic Services Platform: A Components-Oriented Approach. In: V. Getov and T. Kielmann, eds. Component Models and Systems for Grid Applications, Springer-Verlag, 2004.