Preface

Three Institutes of the CoreGRID Network of Excellence organized a common workshop to present and discuss the latest results on Grid and P2P Technologies achieved by the Institute partners as well as the latest developments by researchers outside the CoreGRID community. The three institutes involved are:

- The Institute on *Programming Model*,
- The Institute on *Architectural Issues: Scalability, Dependability, Adaptability*, and
- The Institute on *Grid Systems, Tools, and Environments*.

The aforementioned institutes have a history in organizing joint events – the most recent one being the workshop in Krakow, Poland in June 2006.

The 2007 joint workshop took place at the premises of the Institute of Computer Science, Foundation for Research and Technology-Hellas (FORTH-ICS), Heraklion-Crete, Greece on June 12-13, 2007. The event was organized in two plenary sessions hosting a presentation from the CoreGRID scientific coordinator and three invited speakers, followed by 12 technical sessions hosting the accepted contributions. A total of 42 papers were accepted for presentation among those submitted to the workshop.

The workshop invited speakers presented some of the latest developments in Grid Technology, Semantic Grid, and P2P technology respectively. Their talks focused on ongoing work including some open problems and stimulated interesting discussion among the participants.

The invited speakers and their respective talk titles were the following:

- Dennis Gannon, Indiana University, "*Programming Gateways to the Teragrid*"
- David De Roure, University of Southampton, "*Re- evaluating the Grid*"
- Ann Chervenak, University of Southern California "*Peer-to-peer Approaches to Grid Resource Discovery*".

All the contributions presented to the workshop have been included in a CoreGRID technical report – TR-0080[1]. The papers presented at the workshop have been submitted to a further formal refereeing process. This volume hosts the selected contributions resulting from this review process.

We wish to thank our hosts from FORTH-ICS for their kind hospitality, the local organization team for providing such superb support, the CoreGRID administrative structure for allowing us to organize the workshop and, in particular, to support the invited speakers, and the entire CoreGRID community for submitting such a large number of interesting and high quality papers and for their enthusiastic participation in the workshop.

Our thanks also go to the European Commission for sponsoring this volume of selected articles from the workshop via the CoreGRID NoE project, grant number 004265.

<div style="text-align: right">Marco Danelutto, Paraskevi Fragopoulou, Vladimir Getov</div>

TOWARDS GCM RE-CONFIGURATION - EXTENDING SPECIFICATION BY NORMS

Alessandro Basso, Alexander Bolotov
University of Westminster
Harrow School of Computer Science
University of Westminster
Watford Road
Harrow HA1 3TP

[bassoa.bolotoa]@wmin.ac.uk

Abstract   We continue investigation of formal specification of Grid Component systems by temporal logics and subsequent application of temporal resolution as a verification technique. This time we enrich the specification language by the ability to capture norms which enables us to formally define a concept of a re-configuration. We aim at integrating a software tool for automated specification as well as verification to ensure a reliable and dynamically re-configurable model.

Keywords: formal specification, formal verification, verification tool, GIDE, deductive reasoning, model checking, deontic logic, re-configuration

1.   Introduction.

Component models enable modular design of software applications that 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 there is need for a dynamic re-configuration. The Grid Component Model (GCM) [12]based on Fractal is the one chosen for our research. In these models, components interact together by being bound through interfaces. However, there is a further need for a method which ensures correct composition and behaviour of components. For the specification of behaviour we can use a rich temporal framework [11] with subsequent application of either model checking or deductive reasoning as a verification technique. Model checking [7], which verifies the properties of the components against the specification, has already been tested in various circumstances, one particular application of this method been tested in [2]. Model checking is a powerful and well established technique allowing to incorporate a number of algorithms and tools to deal even with the famous state explosion problem. However, if applied to a component system, it has one indicative drawback, due to its explorative nature, namely, it cannot consider the environment in which a component system has been developed. At the same time, in building a large scale distributed system, we cannot afford anymore not to take into consideration the entire infrastructure. Deductive methods, on the other hand, can deal with such large systems and furthermore, can be applied to re-configuration scenarios. In our earlier work [1] we applied a specific deductive technique, the temporal resolution method [5] to a simple component model. The complexity of the resolution based verification turned out to be high. The analogous method for the linear-time setting has been recently improved in [8] by the modification of the underlying specification language to obtain a polynomial satisfiability checking complexity. In this paper we propose a new framework for the specification of the re-configuration process: the extension of the temporal specification by the deontic modalities [14]. This enriches the expressive capacities of our formal specification by allowing to represent, additionally, a behaviour protocol.

The paper is organised as follows. In §2 we introduce the architecture and give an informal description of the main concepts: re-configuration (§2.1) and model update (§2.2). Further, in §3 we introduce these concepts formally: in §3.1 we describe the deontic extension of $ECTL^+$ called $ECTL_D^+$, then, in §3.2 define a concept of deontic temporal specification (DTS) and reconfiguration, and in §3.3 provide a specification example. Next, in §4 we describe a resolution based verification technique, introducing new deontic resolution rules and providing an example refutation. Finally, in §5, we give concluding remarks and identify future work.

2. Architecture

We identify three main parts of the architecture: the primitive components, their composition into composite components through the Architecture Description Language (ADL) file and the infrastructure (see Figure 1).
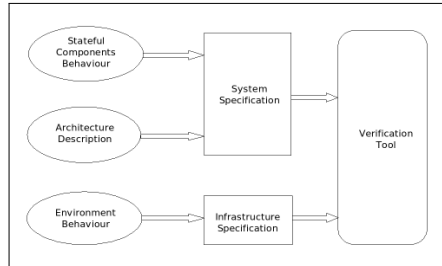

Figure 1. Architecture

The first two are combined to deduce the *stateful component system behaviour* - a high-level behaviour distinct from the one of a single component, which we assume to be already formally verified through other techniques being recently 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. [15] 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 [1], 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.

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 stateful component system behaviour instead, we are taking into consideration a different matter: we are looking for those requirements that will make the component "behaving correctly". As a simple example let us consider a parser which checks if all the libraries required by the component are present to calculate the factorial. Furthermore, 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 should not be stopped? We have taken into consideration these types of situations while developing a specification procedure. We analyse 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 [16]to define a set of states to be generated that would monitored by a specific software [13]. This lifecycle is restricted, in fact it only models the deployment state of the system (and, consequently, the transitions of its states during the life), not its operational characteristics. For example, once a component is in running state, it is 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).

System/Infrastructure Behaviour. To specify the behaviour of a stateful component system, we need, first of all, information on the architecture and hierarchical components structure, the information flow, the possible requirements, and external requests. It is possible to extrapolate interface and bindings information from the XML based ADL file (as similarly outlined in [16]). 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 new parameters. On the other hand, other component's requirements must be taken directly from the component's definitions. Since one of the GridComp functionalities will include a GCM parser to build component models, we will be able to reuse some of the data it will provide to blueprint these requirements, leaving the task to fill in the gaps to a programmer. The infrastructure can represent a general purpose environment based on some common grounds, or a specific one, defined by the programmer. Note that 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 a behaviour protocol of a component system. The requirements of the protocol are understood as norms and specified in terms of deontic modalities, "obligation" and "permission". Note that 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.

2.1   Re-Configuration

We focus our attention on the critical aspect of re-configuration. We begin by clarifying the term re-configuration used in this paper: we refer re-configuration to the process through which a system halts operation under its current source

specification and begins operation under a different target specification [17], and more precisely after the deployment has taken place (dynamic reconfiguration). Some examples include the replacement of a software component by the user, or an automated healing process activated by the system itself. In either of these cases we consider the dynamic reconfiguration process as an unforeseen action at development time (known as ad-hoc reconfiguration [3]). 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 re-configuration procedure is requested or inconsistency detected, the healing process is triggered. The dynamic re-configuration process works in a circular way [Figure: 2] and it is divided into three major steps detailed below. 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.
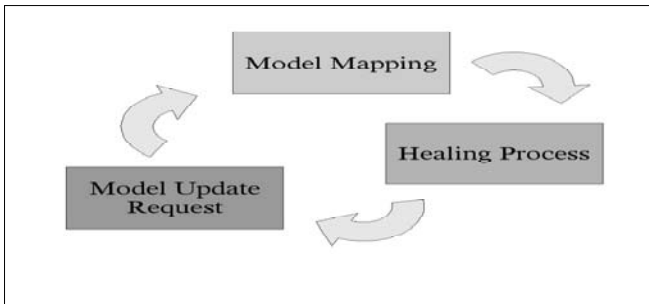

Figure 2.   Re-Configuration Cycle

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 refers 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 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 background process 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 [4].

This process is essential to ensure that no 'past' states are misused by the tool during the healing process.

2.2   Model Update

If the model behaviour needs to be updated according to the new external input parts of the system specification need to be changed. This process is the key to this type of model update architecture and is necessary because, unlike model revision in which the description is simply corrected but the overall system remains unchanged, by updating our specification we are fundamentally changing the system by adding, deleting and replacing states in the model behaviour [9]. Here different types of changes are dealt with in a similar faction, independently from the origin of the update (external user input or self healing process). The behaviour specification is 'extended' to a new type of specification and the verification process is resumed from this point forward [Figure: 3]. This model update process consists of:
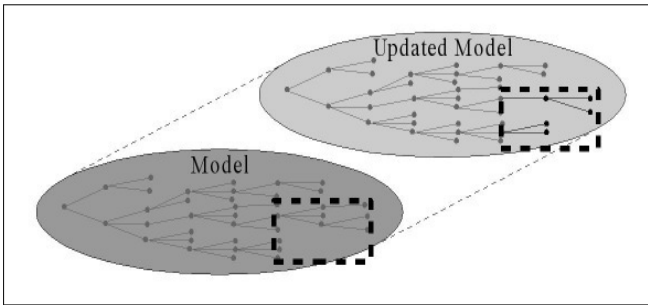

Figure 3.   Model Update

(i) **Norms/Invariant check.** Utilise norms and invariants in the specification for constraints on the set of states to be updated. Here we detect the deontic properties in the specification which could be utilised in the healing process.

(ii) **Compatibility check.** Check if the supplied update to the model conforms with the the set of states to be updated, in other words, the system must check for the presence of the standard bindings of the components, controllers, etc; if so, the model is updated, otherwise, the healing is triggered.

(iii) **Healing process.** Search the tree model for a set of states which conform with the norms and invariants, and is applicable for this set of states. Note that candidate states for such an update in relation for some state s_i, do not have to be in an 'achievable' 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 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. (Note that healing is also triggered if there was no supplied update as in the case of inconsistency detection).

3.   Deontic Extension of the Specification Language

In this paper we introduce a new specification formalism, Temporal Deontic Specification (TDS). 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_{CTL} we are able to subsequently apply the resolution based verification technique which must be also extended to cope with the normative dimension.

Note that 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.

3.1 $ECTL_D^+$ Syntax and Semantic

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: $\square$ – 'always in the future'; $\lozenge$ – '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 'φ is obligatory for an agent a' and $\mathcal{P}_a(\varphi)$ read as 'φ 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)

$S ::= C|S \wedge S|S \vee S|S \Rightarrow S|\neg S|\mathbf{A}P|\mathbf{E}P|\mathcal{P}_aS|\mathcal{O}_aS$
$P ::= P \wedge P|P \vee P|P \Rightarrow P|\neg P|\square S|\lozenge S|\bigcirc S|S\mathcal{U}S|S\mathcal{W}S$
$\qquad |\square\lozenge S|\lozenge\square S$

DEFINITION 1 (LITERAL, MODAL LITERAL) *A* literal *is either p, or ¬p for p is a proposition. A* modal literal *is either $\mathcal{O}_il$, $\neg\mathcal{O}_il$, $\mathcal{P}_il$, $\neg\mathcal{P}_il$ 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 logics.

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*, χ_{s_i} 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 χ be a family of all paths of $\mathcal{M}$. A path $\chi_{s_0} \in \chi$ is called a *fullpath*. Let X be a family of all fullpaths of $\mathcal{M}$. Given a path χ_{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 χ_{s_i} *a prefix* of a path χ_{s_i} and an infinite sub-sequence $s_j, s_{j+1}, s_{j+2} \ldots$ of χ_{s_i} *a suffix* of a path χ_{s_i} abbreviated $Suf(\chi_{s_i}, s_j)$.

Following [11], without loss of generality, we assume that underlying tree models are of at most countable branching.

DEFINITION 3 (TOTAL COUNTABLE ω-TREE) *A countable ω-tree, τ_ω, 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 [11], 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 ω 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 countable total 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 ω-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 [11], in Figure 4 we define a relation '$\models$', which evaluates well-formed ECTL_D^+ formulae at a state s_m in a model $\mathcal{M}$.

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

$$
\begin{aligned}
&\langle \mathcal{M}, s_m \rangle &&\models p &&\text{iff} &&p \in L(s_m), \text{ for } p \in Prop \\
&\langle \mathcal{M}, s_m \rangle &&\models \mathbf{A}B &&\text{iff} &&\text{for each } \chi_{s_m}, \langle \mathcal{M}, \chi_{s_m} \rangle \models B \\
&\langle \mathcal{M}, s_m \rangle &&\models \mathbf{E}B &&\text{iff} &&\text{there exists } \chi_{s_m} \text{ such that} \\
& && && &&\langle \mathcal{M}, \chi_{s_m} \rangle \models B \\
&\langle \mathcal{M}, \chi_{s_m} \rangle &&\models A &&\text{iff} &&\langle \mathcal{M}, s_m \rangle \models A, \text{ for state formula } A \\
&\langle \mathcal{M}, \chi_{s_m} \rangle &&\models \square B &&\text{iff} &&\text{for each } s_n \in \chi_{s_m}, \text{ if } m \le n \text{ then} \\
& && && &&\langle \mathcal{M}, Suf(\chi_{s_m}, s_n) \rangle \models B \\
&\langle \mathcal{M}, \chi_{s_m} \rangle &&\models \bigcirc B &&\text{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 &&\text{iff} &&\text{there exists } s_n \in \chi_{s_m} \text{ such that } m \le n \\
& && && &&\text{and } \langle \mathcal{M}, Suf(\chi_{s_m}, s_n) \rangle \models B \\
& && && &&\text{and for each } s_k \in \chi_{s_m}, \text{ if } m \le k < n \\
& && && &&\text{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 &&\text{iff} &&\langle \mathcal{M}, \chi_{s_m} \rangle \models \square A \text{ or } \langle \mathcal{M}, \chi_{s_m} \rangle \models A\,\mathcal{U}\,B \\
&\langle \mathcal{M}, s_m \rangle &&\models \mathcal{O}_a B &&\text{iff} &&\text{for each } s_n \in S, \text{ if } D_a(m, n) \text{ then} \\
& && && &&\langle \mathcal{M}, s_n \rangle \models B \\
&\langle \mathcal{M}, s_m \rangle &&\models \mathcal{P}_a B &&\text{iff} &&\text{there exists } s_n \in S, \text{ such that } D_a(m, n) \\
& && && &&\text{and } \langle \mathcal{M}, s_n \rangle \models B
\end{aligned}
$$

Figure 4. ECTL$_D^+$ semantics

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

3.2 Reconfiguration formalisation

To define a concept of propositional deontic temporal specification we extend a normal form defined for the logic ECTL$^+$, SNF$_{\text{CTL}}$, which was developed in [5–6]. Recall that the core idea of the normal form is to extract from a given formula the following three types of constraints. *Initial constraints* represent information relevant to the initial moment of time, the root of a tree. *Step constraints* of the form indicate what will happen at the successor state(s) given that some conditions are satisfied 'now'. Finally, *sometime constraints* keep track on any eventuality, again, given that some conditions are satisfied 'now'.

The SNF$_{\text{CTL}}^D$ language is obtained from the ECTL$_D^+$ language by omitting the $\mathcal{U}$ and $\mathcal{W}$ operators, and adding classically defined constants **true** and **false**, and a new operator, **start** ('at the initial moment of time') defined as $\langle \mathcal{M}, s_i \rangle \models \mathbf{start}$ iff $i = 0$.

Similarly to SNF$_{\text{CTL}}$, we incorporate the language for indices which is based on the set of terms IND $= \{\langle \mathsf{f} \rangle, \langle \mathsf{g} \rangle, \langle \mathsf{h} \rangle, \langle LC(\mathsf{f}) \rangle, \langle LC(\mathsf{g}) \rangle, \langle LC(\mathsf{h}) \rangle \ldots \}$, where f, g, h ... denote constants. Thus, $\mathbf{E}A_{\langle \mathsf{f} \rangle}$ means that A holds on some path labelled as $\langle \mathsf{f} \rangle$. All formulae of SNF$_{\text{CTL}}$ of the type $P \Rightarrow \mathbf{E}\bigcirc Q$ or

$P \Rightarrow \mathbf{E}\Diamond Q$, where Q is a purely classical expression, are labelled with some index.

DEFINITION 7 (DEONTIC TEMPORAL SPECIFICATION - DTS) *DTS 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 α_i, β_m, γ 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, and $\langle \mathsf{ind} \rangle \in \mathsf{IND}$ is some index.*

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

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

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

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

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

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

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

In order to give a formalisation of the reconfiguration process we adapt the approach given in [17] extending it to the usage of norms. We assume that we are given a set of specification properties, S_i be the start state and S_j the end state of the system, a set of norms, N, and a set of invariants I. We can define a reconfiguration, $\mathcal{R}$, to be applicable when the following conditions holds:

- $\mathcal{R}$ commences when the initial state S_i is not operating anymore and finishes before the last state to be updated, S_j, becomes compliant with the system.
- S_j is the appropriate choice for the target specification at some point during $\mathcal{R}$.
- Time for $\mathcal{R}$ is less or equal than the time for the transition from S_i to S_j.
- The transition invariant(s), I, holds during $\mathcal{R}$.
- The norms, N, for S_j are true at the time when $\mathcal{R}$ finishes.
- The lifetime of $\mathcal{R}$ is bounded by any two occurrences of the same specification.

The conditions for reconfigurations can be considered as a set of restriction, which when true allow for the model to be simply replaced. The reconfiguration conditions above give a clear indication to which states in the model can be changed and when, while the temporal specification sets the conditions for the change and defines the acceptable states which will replace the current ones.

3.3   Example Specification

Let us consider an example specification in which we use of norms for reconfiguration, and where a component is requested to be updated.

Let r represent a property that a core component is bound to the system (one that should be always available and should not be 'touched'), and let q be a new upgraded version of this core component. Now the expression $\mathbf{A}\,\square\,(r \Rightarrow \mathcal{O}_i \neg q)$ stands for the obligation of not binding this new component once r is present.

Assume that the system received a request for the permission to eventually bind q. 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 DTS clauses).

Conditions of the System	Constraints of DTS
Invariant Property r	$\mathbf{A}\Box((\mathbf{start} \Rightarrow r) \wedge (r \Rightarrow \mathbf{A}\bigcirc r))$
Obligation of not binding new component q	$\mathbf{A}\Box(r \Rightarrow \mathcal{O}_i\neg q)$
A request for the permission to eventually bind q	$\mathbf{A}\Box((r \Rightarrow \mathbf{E}\Diamond w) \wedge (w \Rightarrow \mathcal{P}_i q))$

4. Resolution Based Verification Technique

We first update the set of resolution rules developed for SNF_{CTL} [6] 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.

$$
\begin{array}{lll}
DRES & SRES & TRES \\
\mathbf{true} \Rightarrow D \vee O_i l & A \Rightarrow \mathbf{A}\bigcirc(l \vee D) & A \Rightarrow \mathbf{A}\Box l \\
\mathbf{true} \Rightarrow D' \vee P_i \neg l & B \Rightarrow \mathbf{A}\bigcirc(\neg l \vee E) & B \Rightarrow \mathbf{E}\Diamond l_{\langle f \rangle} \\
\hline
\mathbf{true} \Rightarrow D \vee D' & A \wedge B \Rightarrow \mathbf{A}\bigcirc(D \vee E) & B \Rightarrow \mathbf{E}(\neg A \, \mathcal{W} \, l)_{\langle f \rangle}
\end{array}
$$

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

TDS		$Proof$	
1. $\mathbf{start} \Rightarrow r$	6.	$\mathbf{true} \Rightarrow \neg r \vee \mathcal{O}_i \neg q$	$from\ 3$
2. $\quad r \Rightarrow \mathbf{A}\bigcirc r$	7.	$\mathbf{true} \Rightarrow \neg w \vee \mathcal{P}_i q$	$from\ 5$
3. $\quad r \Rightarrow \mathcal{O}_i \neg q$	8.	$\mathbf{true} \Rightarrow \neg r \vee \neg w$	$DRES,\ 6,7$
4. $\quad r \Rightarrow \mathbf{E}\Diamond w_{\langle f \rangle}$	9.	$\mathbf{start} \Rightarrow \neg r \vee \neg w$	$temporising,\ 8$
5. $\quad w \Rightarrow \mathcal{P}_i q$	10.	$\mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg r \vee \neg w)$	$temporising,\ 8$
	11.	$\quad r \Rightarrow \mathbf{A}\bigcirc w$	$SRES,\ 2,10$
	12.	$\quad r \Rightarrow \neg r \, \mathcal{W} \, w$	$TRES,\ 2,11,4$
	13.	$\quad r \Rightarrow w \vee \neg r$	$\mathcal{W}\ removal,\ 12$
	14.	$\mathbf{start} \Rightarrow \neg r \vee w$	$\mathcal{W}\ removal,\ 12$
	15.	$\mathbf{start} \Rightarrow \mathbf{false}$	$SRES,\ 1,9,14$

Here the reconfiguration request is rejected, hence no changes to the model.

5. Conclusions

The need for a safe and reliable way to reconfigure systems, especially distributed, resource depending and long running systems, has led to the need for a formal way to describe and verify them before risking to take some action. In

this paper we have given a definition for dynamic reconfiguration and a formal way to specify the behaviour of a component model and its infrastructure. Furthermore we have demonstrated how we can apply this formal specification to model update techniques that conform to the definition of dynamic reconfiguration given. The method introduced can also be used to prevent inconsistency and suggest corrections to the system in a static and/or dynamic environment. Indeed, if the verification technique discovers inconsistencies in the configuration then the 'healing' process is triggered: the process of "re-configuring" of the computation tree model that conforms the protocol. To ensure the consistency we must supply the system with internal 'clocks' which is needed to synchronise the states that belong to different branches of the computation tree. We are planning to eventually embed all these features in a prototype plug-in for the GridComp GIDE and test it on case studies proposed by industry partners.

References

[1] 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.

[2] T. Barros and L. Henrio and E. Madelaine. Verification of Distributed Hierarchical Components. In *Proc. of the International Workshop on Formal Aspects of Component Software (FACS'05)*. Electronic Notes in Theor. Computer Sci. 160. pp. 41-55 (ENTCS), 2005.

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

[4] F. Baude and D. Caromel and F. Huet and L. Mestre and J. Vayssi 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 p. 93, IEEE Computer Society.

[5] A. Bolotov. *Clausal Resolution for Branching-Time Temporal Logic.* PhD thesis, Department of Computing and Mathematics, The Manchester Metropolitan University, 2000.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] M. Endler. A language for implementing generic dynamic reconfigurations of distributed programs. In Proceedings of the 12th Brazilian Symposium on Computer Networks, pages 175-187, 1994.

[11] 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.

[12] GCM program committee Basic Features of the Grid Component Model Deliverable D.PM.04, CoreGRID, March 2007.

[13] V. Getov, A. Basukoski, J. Thiyagalingam, Y. Yulai and Y. Wu Grid programming with COMPonents : an advanced component platform for an effective invisible grid GRIDComp Technical Report, July 2007

[14] 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.

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

[16] S. Schaefer. CDDLM - Component Model In proceeding to the Open Grid Forum, March 2006

[17] Elisabeth A. Strunk and John C. Knight. Assured Reconfiguration of Embedded Real-Time Software. DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04), 2004, p. 367, IEEE Computer Society.