Specification and verification of reconfiguration protocols in grid component systems.

Alessandro Basso[1]
Alexander Bolotov[1]
Artie Basukoski[1]
Vladimir Getov[1]
Ludovic Henrio[2]
Mariusz Urbanski[3]

[1] Harrow School of Computer Science
[2] CNRS/I3S/INRIA, Sophia Antipolis, France.
[3] Institute of Psychology, Adam Mickiewicz University, Poznan, Poland.

Specification and Verification of Reconfiguration Protocols in Grid Component Systems

Alessandro Basso[1], Alexander Bolotov[1], Artie Basukoski[1], Vladimir Getov[1], Ludovic Henrio[2] and Mariusz Urbanski[3]

Abstract—
In this work we present an approach for the formal specification and verification of the reconfiguration protocols in Grid component systems. We consider Fractal, a modular and extensible component model. As a specification tool we invoke a specific temporal language, separated clausal normal form, which has been shown to be capable of expressing any ECTL$^+$ expression, thus, we are able to express the complex fairness properties of a component system. The structure of the normal enables us to directly apply the deductive verification technique, temporal resolution defined in the framework of branching-time temporal logic.

I. 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 [30] which can be brought on-line whenever necessary [26].

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 and verification of the core component model and the properties of the desired Grid systems.

Among various approaches to representing a component model we pay specific attention to the Fractal component model [17]. The advantage of the Fractal framework is that it defines the structure of the components, gives a basic classification of components, and has the mathematical foundations, e.g., the Kell calculus [7]. The Fractal specification defines the basic (non-functional) controls which should be defined especially to enable dynamic reconfiguration of components, and a number of constraints on the interplay between functional and non-functional operations. The reconfiguration is obtained by triggering appropriate actions on specific types of the components' interfaces. These explicit dynamic properties of the Fractal component model are particularly suitable for Grid systems and environments. In this work we focus on predefined categories of reconfigurations and also on proving properties of these reconfigurations.

Given a formal specification of a distributed system there are two major approaches to formal verification of this specification: algorithmic and deductive [28]. 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.

In our approach the components are modelled in a specific branching-time temporal logic, or $\mathrm{SNF}_{\mathrm{CTL}}$(Separated Normal Form for Computation Tree Logic) [14], and then the temporal resolution is applied as a deductive verification tool.

$\mathrm{SNF}_{\mathrm{CTL}}$ initially developed for CTL has been shown to be able to express simple fairness constraints and their Boolean combinations [10], [11]. Furthermore, a clausal resolution over the set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses has been defined [9], [10] and recently the search strategies for this method were presented in [12].

These developments allow us to set up the following problem structure tackled in the paper:

$$FCM \longrightarrow SNF_{ctl}(FCM) \longrightarrow BTR$$

Here we suggest the translation of the Fractal component model (FCM) into the $\mathrm{SNF}_{\mathrm{CTL}}$(FCM), the $\mathrm{SNF}_{\mathrm{CTL}}$ based formal specification of FCM, and to apply the 'branching temporal resolution' method (BTR), the temporal resolution technique defined over the set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses.

II. Reconfiguration Scenario

A. Component Model

Fractal is a modular and extensible component model. The Fractal specification defines a set of notions characterizing this model, an API (Application Program Interface), and an ADL (Architecture Description Language).

Components are characterized 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-

[1] Harrow School of Computer Science, University of Westminster, London, UK.
[2] CNRS/I3S/INRIA, Sophia Antipolis, France.
[3] Institute of Psychology, Adam Mickiewicz University, Poznan, Poland.
This work was partially supported by the FP6 Network of Excellence CoreGRID funded by the European Commission under contract IST-2002-004265.

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.

Fractal is a multi-level specifications. Depending on their conformance level, Fractal components can feature introspection and/or configuration. The *control* interfaces are used in the Fractal 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 functionalities. A *functional* interface can provide the required functionalities and we call it the *server* interface. Alternatively, a *client* interface requires some other functionalities.

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

There are four controllers that have been already defined in Fractal (but others may be user-defined depending on the needs of the model):

- 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; note that 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.

These are the basic controls which should be defined especially to be able to have dynamic reconfiguration of components.

The Fractal specification defines a number of constraints on the interplay between functional and non-functional operations:

- Content and binding control operations are only possible when the component is stopped.
- When stopped, a component does not emit invocations and must accept invocations through control interfaces; whether or not an invocation to a functional interface is possible is undefined.

B. Configuration/Reconfiguration Scenario

In general, the initial configuration of a Fractal component is given by the description of the component using Fractal ADL.

From this first state, reconfiguration is obtained by triggering appropriate actions on the the life-cycle, the binding, and the content control interfaces. A reconfiguration can be triggered by any component that has a reference to a correct non-functional interface.

In this work we focus on predefined categories of reconfigurations and on proving properties on these reconfiguration. As far as the reconfiguration is concerned we use the classical assumption that replacing a component by a similar one is safe for the system.

III. Specification of the Scenario in Temporal Logic Framework

A. Formal Specification and Verification of Components

We distinguish the specification of the primitives and of the composite component. The primitives are specified as a black box, usually in a programming language of our choice. The component composition is specified using Fractal ADL, and from this specifications it is possible to extract the bindings between interfaces of subcomponents and the controllers of the component itself.

B. Specification Language: Normal Form for ECTL$^+$

The language of a normal form, SNF_{CTL} developed for a number of branching-time logics, CTL [8], [14], ECTL [10] and ECTL$^+$ [11], is based upon the extended set of classical logic operators $\wedge$, $\vee$, $\Rightarrow$, $\neg$, the set of future time temporal operators $\Box$ (always), $\Diamond$ (sometime), $\bigcirc$ (next time) and path quantifiers $\mathbf{A}$ (on all future paths) and $\mathbf{E}$ (on some future path), classically defined constants **true** and **false**, and a new operator, **start** ('at the initial moment of time' with the intended meaning that it is true only at the initial moment of time).

Underlying Tree Structures. Assuming familiarity of the reader with the basic tree structure concepts we precede the presentation of the SNF_{CTL} language by the introduction of the notation that we will utilise.

Definition 1 (Tree) A *tree*, $\mathcal{T}$, 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 (a) $s_0 \in S$ is a unique root node (b)for every $s_i \in S$ there exists $s_j \in S$ such that $R(s_i, s_j)$; and (c) for every s_i, s_j, $s_k \in S$, if $R(s_i, s_k)$ and $R(s_j, s_k)$ then $s_i = s_j$.

By χ_{s_i} we abbreviate a path departing from s_i. A path χ_{s_0} is called a *fullpath*. Let X be a family of all fullpaths of $\mathcal{T}$. 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 $Suf(\chi_{s_i}, s_j)$.

We assume that the underlying trees are countable ω-trees, i.e. any fullpath $\chi \in X$ is isomorphic to natural numbers and every state $s_i \in S$ has a countable number of successors.

Now we are ready to define the formal syntax and semantics for SNF_{CTL}. A set of SNF_{CTL} clauses is interpreted in a structure $\mathcal{M} = \langle S, R, s_0, X, L \rangle$, where (S, R) is a countable ω tree with a root s_0, X is a set of all fullpaths and L is an interpretation function mapping atomic propositional symbols to truth values at each state and the following condition is satisfied: X is R-generable [19], i.e. for every state $s_i \in S$, there exists $\chi_j \in X$ such that $s_i \in \chi_j$, and for every sequence $\chi_j = s_0, s_1, s_2, \ldots$, the following is true: $\chi_j \in X$ if, and only if, for every i, $R(s_i, s_{i+1})$.

In our definition of an $\mathrm{SNF_{CTL}}$ model structure $\mathcal{M}$ the set of fullpaths X is R-generable. Therefore, following [19], it it is suffix, fusion and limit closed.

Syntax. First, we fix a countable set, $Prop = x, y, z, \ldots$, of atomic propositions. The core idea of $\mathrm{SNF_{CTL}}$ is to represent temporal information in the following three types of constraints. *Initial constraints* represent information relevant to the initial moment of time, the root of the computation tree. *Step constraints* 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'. Additionally, to enable sound reasoning within a specific path context during the verification, we incorporate indices.

Indices. The language for indices is based on the set of terms $\mathsf{IND} = \{\langle \mathsf{f} \rangle, \langle \mathsf{g} \rangle, \langle \mathsf{h} \rangle, \langle LC(\mathsf{f}) \rangle, \langle LC(\mathsf{g}) \rangle, \langle LC(\mathsf{h}) \rangle \ldots\}$, where f, g, $\mathsf{h}\ldots$ denote constants. Thus, $\mathbf{E}A_{\langle \mathsf{f} \rangle}$ means that A holds on some path labelled as $\langle \mathsf{f} \rangle$. A designated type of indices in $\mathrm{SNF_{CTL}}$ are indices $\langle \mathsf{LC(ind)} \rangle$ which represent a limit closure of prefixes associated with $\langle \mathsf{ind} \rangle$. All Formulae of $\mathrm{SNF_{CTL}}$ of the type $P \Rightarrow \mathbf{E}\bigcirc Q$ or $P \Rightarrow \mathbf{E}\Diamond Q$, where Q is a purely classical expression, are labeled with some index.

Definition 2 (Separated Normal Form $\mathrm{SNF_{CTL}}$) A set of $\mathrm{SNF_{CTL}}$ clauses is a set of Formulae $\mathbf{A}\Box[\ _i(P_i \Rightarrow F_i)]$ where each of the *clauses* $P_i \Rightarrow F_i$ is further restricted as below, each $\alpha_j, \alpha_p, \alpha_t, \alpha_v, \beta_i, \beta_m, \beta_r$ or γ is a literal, **true** or **false** and $\langle \mathsf{ind} \rangle \in \mathsf{IND}$ is some index.

start	$\Rightarrow$	$_{i=1}^{k}\beta_i$	an initial clause
$_{j=1}^{l}\alpha_j$	$\Rightarrow$	$\mathbf{A}\bigcirc[\ _{m=1}^{n}\beta_m]$	an **A** step clause
$_{p=1}^{q}\alpha_p$	$\Rightarrow$	$\mathbf{E}\bigcirc[\ _{r=1}^{s}\beta_r]_{\langle \mathsf{ind} \rangle}$	an **E** step clause
$_{t=1}^{u}\alpha_t$	$\Rightarrow$	$\mathbf{A}\Diamond\gamma$	an **A** sometime clause
$_{v=1}^{w}\alpha_v$	$\Rightarrow$	$\mathbf{E}\Diamond\gamma_{\langle \mathsf{LC(ind)} \rangle}$	an **E** sometime clause

C. *Interpreting $\mathrm{SNF_{CTL}}$.*

We define a relation $\models$ which evaluates the $\mathrm{SNF_{CTL}}$ clauses at a state s_i in a model $\mathcal{M}$. The evaluation of the classical connectives in the states is standard. Below we represent the evaluation of the temporal operators and path quantifiers.

$\langle \mathcal{M}, s_i \rangle \models \mathbf{A}B$ iff for each χ_{s_i}, $\langle \mathcal{M}, \chi_{s_i} \rangle \models B$.
$\langle \mathcal{M}, s_i \rangle \models \mathbf{E}B$ iff there exists χ_{s_i} such that $\langle \mathcal{M}, \chi_{s_i} \rangle \models B$.
$\langle \mathcal{M}, \chi_{s_i} \rangle \models \Box B$ iff for each $s_j \in \chi_{s_i}$, if $i \leq j$ then $\langle \mathcal{M}, Suf(\chi_{s_i}, s_j) \rangle \models B$.
$\langle \mathcal{M}, \chi_{s_i} \rangle \models \Diamond B$ iff there exists $s_j \in \chi_{s_i}$ such that $i \leq j$ and $\langle \mathcal{M}, Suf(\chi_{s_i}, s_j) \rangle \models B$.
$\langle \mathcal{M}, \chi_{s_i} \rangle \models \bigcirc B$ iff $\langle \mathcal{M}, Suf(\chi_{s_i}, s_{i+1}) \rangle \models B$.

Definition 3 (Satisfiability, validity) An $\mathrm{SNF_{CTL}}$ clause, C, is satisfiable if, and only if, there exists a model $\mathcal{M}$ such that $\langle \mathcal{M}, s_0 \rangle \models C$. An $\mathrm{SNF_{CTL}}$ clause, C, is valid if, and only if, it is satisfied in every possible model.

An initial $\mathrm{SNF_{CTL}}$ clause, **start** $\Rightarrow F$, is understood as "F is satisfied at the initial state of some model $\mathcal{M}$". Any other $\mathrm{SNF_{CTL}}$ clause is interpreted taking also into account that it occurs in the scope of $\mathbf{A}\Box$.

Thus, a clause $\mathbf{A}\Box(x \Rightarrow \mathbf{A}\bigcirc p)$ is interpreted as "*for any fullpath χ and any state $s_i \in \chi$ $(0 \leq i)$, if x is satisfied at a state s_i then p must be satisfied at the moment, next to s_i, along each path which starts from s_i*".

Next, a clause $\mathbf{A}\Box(x \Rightarrow \mathbf{E}\bigcirc q_{\langle \mathsf{ind} \rangle})$ is interpreted as "*for any fullpath χ and any state $s_i \in \chi$ $(0 \leq i)$, if x is satisfied at a state s_i then q must be satisfied at the moment, next to s_i, along a path which starts from s_i and which is associated with* ind". Speaking informally, we interpret $\mathbf{A}\Box(x \Rightarrow \mathbf{E}\bigcirc q_{\langle \mathsf{ind} \rangle})$ such that given a state in a model which satisfies x (the left hand side of the clause), the label, ind, indicates the direction, in which the successor state which satisfies q can be reached (see similar developments in the construction of logic DCTL$*$ [27]).

Finally, we would like to point out that our interpretation of an LC index corresponds to the concept of a linear interpretation [36].

Note that in the full $\mathrm{ECTL^+}$ language the standard 'until' ($\mathcal{U}$) and 'unless' ($\mathcal{W}$) operators are used:
$\langle \mathcal{M}, \chi_{s_i} \rangle \models A\,\mathcal{U}\,B$ iff there exists $s_j \in \chi_{s_i}$ such that $i \leq j$ and $\langle \mathcal{M}, Suf(\chi_{s_i}, s_j) \rangle \models B$ and for each $s_k \in \chi_{s_i}$, if $i \leq k < j$ then $\langle \mathcal{M}, Suf(\chi_{s_i}, s_k) \rangle \models A$.
and $A\,\mathcal{W}\,B = \Box A \vee A\,\mathcal{U}\,B$. In the $\mathrm{SNF_{CTL}}$ these operators are defined via the basic set of $\mathrm{SNF_{CTL}}$ operators [8]. For example, the following rules can be applied to remove the $\mathcal{W}$ operator in the scope of either of the path quantifiers [8] where x is a new proposition:

Removal of $\mathbf{E}\mathcal{W}$	**Removal of** $\mathbf{A}\mathcal{W}$
$P \Rightarrow \mathbf{E}(p\,\mathcal{W}\,q)_{\langle LC(\mathsf{ind}) \rangle}$	$P \Rightarrow \mathbf{A}(p\,\mathcal{W}\,q)$
$P \Rightarrow q \vee (p \wedge x)$	$P \Rightarrow q \vee (p \wedge x)$
$x \Rightarrow \mathbf{E}\bigcirc(q \vee (p \wedge x))_{\langle \mathsf{ind} \rangle}$	$x \Rightarrow \mathbf{E}\bigcirc(q \vee (p \wedge x))$

D. *Example Specification*

Let us consider a simple printing queue component model which consists of a client and one printing queue component as primitives. The client interfaces of the client are of type CI_a and the server interfaces of the printing queue are of type SI_r. Finally, we have a simplified version of a life-cycle controller that allows to safely add or remove a binding between a client and the printing queue.

Formal specification of non-functional aspects. In order to allow for reconfiguration, not only the scenario must be formally specified, but also everything else which allows dynamic reconfiguration. Although in the fractal model four controller interfaces are defined, for reasons of space, we will only specify the safe-unbinding part of a reduced Life-Cycle Controller (LCC) so that it can be used in the deductive reasoning. Note that it is always possible to create new controllers if needed, in this case an appropriate set of formal specifications for each controller must be provided using a similar procedure. If a controller follows the standard Fractal model, a standard set of general temporal logic rules can be called and then modified to match the specification; otherwise, in the case of user-made definitions, the programmers themselves must provide the rules matching the criteria followed in the creation of the definition.

Next we will let the propositions $Bound_1, \ldots, Bound_n$ denote the bindings between components. The format that each may take is $Bound_i(CI_a, SI_r)$ $(1 \leq i \leq n)$ which is a proposition that (when true) specifies that a component with Client Interface CI_a is bound to the Server Interface SI_r. In this example we have two primitive components, one for the Printing Queue and one for the Client using the Printing Queue. We would add as many of these propositions as necessary to describe the system.

LCC is a proposition which when true signifies that the Life Cycle Controller is active.

Before introducing the Life Cycle Controller Formula we would need to specify how components are started and stopped. However, for illustration in the context of this paper we will only provide a partial specification of the Life Cycle Controller and two primitive components; we only deal with the formula that captures the bindings of the two components. We will model the start of the components by attaching them to **start**.

Now we introduce the formula for our version of the Simplified Life-Cycle Controller:

$$\neg LCC \wedge \neg(Bound_1(CI_a, SI_r) \vee Bound_2(CI_a, SI_r))$$

$$\Rightarrow \mathbf{A}\,\Box LCC \Rightarrow (Bound_1(CI_a, SI_r) \wedge Bound_2(CI_a, SI_r))$$

which states that if neither of the components are bound and the LCC is not active then in all possible computations when the LCC is active then we must have the two components bound.

In the following example the Client can send a request for printing: $req(CI_a)$ abbreviated below as req. When true, this proposition states that a printing request has been raised by the client which possesses the client interface CI_a. Similarly $print$ is a proposition stating that a printing request has been satisfied by the printer.

Formal specification of reconfiguration scenarios. For this section, we consider a simple printing queue component model (see figure 1) which consists of one client and one printing queue component as primitives. The client interface of the client is labelled CI_a, and the server interfaces of the printing queue is labelled SI_r. We will also consider a simplified life-cycle controller LCC that allows us to safely remove a binding between a client and the printing queue. This simple example is sufficient to demonstrate the potential of deductive reasoning, applied to a fractal model.

We will take in consideration the safety part of the specification and its requirements [29]. The Life-Cycle Controller LCC does not have a set specification being a non-functional component. We suggest that the system has

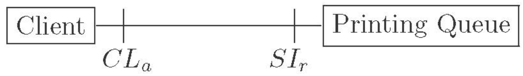

Fig. 1. Example in Fractal: LC Controller

a common protocol of communication (both Client and Printing Queue must follow a common process when a request is raised).

Client specification:

(1) $req \Rightarrow \mathbf{A}(req\,\mathcal{U}\,(req \wedge print))$	Request is kept until it is possible to execute it
(2) $req \Rightarrow \mathbf{A}(\neg req2\,\mathcal{U}\,print)$	There will be no other request until job is printed
(3) $req \Rightarrow \mathbf{A}\Diamond\neg req$	The request for print will be eventually released

The complete specification of the primitive:

$$\mathbf{start} \Rightarrow \neg req \wedge (1) \wedge (2) \wedge (3)$$

where $\neg req$ defines the initial state for Client primitive.

Printing queue specification:

(4)$\mathbf{A}\,\Box\neg(print \wedge print2)$	Mutual Exclusion property: at every point in time, the printer can perform at most one printing operation:
(5) $\mathbf{A}(\neg print\,\mathcal{W}\,req)$	There is no printing unless requested
(6) $print \Rightarrow \mathbf{A}\Diamond\neg print$	Printing will eventually end
(7) $req \Rightarrow \mathbf{A}\Diamond print$	The request for printing should be granted

The complete specification of the primitive:

$$\mathbf{start} \Rightarrow \neg print \wedge (4) \wedge (5) \wedge (6) \wedge (7)$$

Finally we specify the Life-Cycle Controller properties which affect the receiving of a printing request and the printing itself:

$\mathbf{start} \Rightarrow [(\neg LCC \wedge \neg(req \vee print)) \Rightarrow \mathbf{A}\,\Box(LCC \Rightarrow (req \wedge print))]$

When the life-cycle controller is activated, it ensures that Client Interface and Server Interface are bound, therefore allowing for requests to be sent from the Client, and prints to be carried out by the Printing Queue, for the specific binding.

We believe that the branching-time framework is appropriate for our specification targets because of the following reasons. Assume that after unbinding a client CI, it has been removed forever. Now, from this moment of time it is true to say that $\mathbf{A}\,\Box\neg req$ (in all possible futures from now on, there will be no more requests from the Client Interface to the Server Interface) and therefore at the previous moment of time it was true to say that $\mathbf{E}\bigcirc\mathbf{A}\,\Box\neg req$ (in some future it will not be possible for the Client interface to send a request to the Server Interface). The branching-time framework used shows how significant its use can be even in such simple example.

To apply deductive reasoning to this model, various properties could be taken into consideration. As a relatively simple example we consider the following property. Let p stands for $\neg req(CI_a, SI_r) \wedge \neg print(CI_a, SI_r)$. Assume now that during the reconfiguration of the system the following property should be verified:

$$\ddagger \qquad \mathbf{A}(\Box\Diamond p \wedge \Diamond\Box\neg p)$$

In the next section we will show how this formula can be represented in terms of $\mathrm{SNF}_{\mathrm{CTL}}$ and then apply to this specification the resolution technique as a verification method.

IV. The Verification Method - Clausal Temporal Resolution

A. Temporal Resolution Method for Branching-Time Logics

In order to achieve a refutation of the generated specification, we incorporate two types of resolution rules already defined in [8], [14]: *step* resolution (SRES) and *temporal* resolution (TRES). Step resolution is used between Formulae that refer to the *same* initial moment of time or *same* next moment along some or all paths. Two step resolution rules that will be used in our example are given below (where l is a literal and C and D are disjunctions of literals).

SRES 1

$$\frac{\begin{array}{l}\mathbf{start} \Rightarrow C \vee l \\ \mathbf{start} \Rightarrow D \vee \neg l\end{array}}{\mathbf{start} \Rightarrow C \vee D}$$

SRES 3

$$\frac{\begin{array}{r}P \Rightarrow \mathbf{A}\bigcirc(C \vee l) \\ Q \Rightarrow \mathbf{E}\bigcirc(D \vee \neg l)_{\langle\mathsf{ind}\rangle}\end{array}}{(P \wedge Q) \Rightarrow \mathbf{E}\bigcirc(C \vee D)_{\langle\mathsf{ind}\rangle}}$$

When an empty constraint is generated on the right hand side of the conclusion of the resolution rule, we introduce a constant **false** to indicate this terminating clause.

Now we present only two temporal resolution rules that will be used in our verification example. In the formulation of these rules below l is a literal and the first premises abbreviate the **A** and **E** loops in l [13], i.e. the situation where, given that P is satisfied at some point of time, l occurs always from that point on all or some path respectively.

TRES 2

$$\frac{\begin{array}{l}P \Rightarrow \mathbf{A}\bigcirc\mathbf{A}\Box l \\ Q \Rightarrow \mathbf{E}\Diamond\neg l_{\langle LC(\mathsf{ind})\rangle}\end{array}}{Q \Rightarrow \mathbf{E}(\neg P\, \mathcal{W} \neg l)_{\langle LC(\mathsf{ind})\rangle}}$$

TRES 3

$$\frac{\begin{array}{l}P \Rightarrow \mathbf{E}\bigcirc\mathbf{E}\Box l_{\langle LC(\mathsf{ind})\rangle} \\ Q \Rightarrow \mathbf{A}\Diamond\neg l\end{array}}{Q \Rightarrow \mathbf{A}(\neg P\, \mathcal{W} \neg l)}$$

B. Example Verification

To verify ($\ddagger$) we apply the resolution method to the set of $\mathrm{SNF}_{\mathrm{CTL}}$ clauses $\mathrm{SNF}_{\mathrm{CTL}}(\ddagger)$. We commence the resolution proof presenting at steps 1 – 13 the clauses of $\mathrm{SNF}_{\mathrm{CTL}}(\ddagger)$ in the following order: initial clauses, step clauses and, finally, any sometime clauses.

1.	$\mathbf{start} \Rightarrow x$	8.	$x_1 \Rightarrow \mathbf{A}\bigcirc y$
2.	$\mathbf{start} \Rightarrow \neg x \vee y$	9.	$x_1 \Rightarrow \mathbf{A}\bigcirc x_1$
3.	$\mathbf{start} \Rightarrow \neg x \vee x_1$	10.	$z_1 \Rightarrow \mathbf{E}\bigcirc\neg p_{\langle\mathsf{f}\rangle}$
4.	$\mathbf{start} \Rightarrow \neg z \vee \neg p$	11.	$z_1 \Rightarrow \mathbf{E}\bigcirc z_{1\langle\mathsf{f}\rangle}$
5.	$\mathbf{start} \Rightarrow \neg z \vee z_1$	12.	$y \Rightarrow \mathbf{A}\Diamond p$
6.	$\mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg z \vee \neg p)$	13.	$x \Rightarrow \mathbf{E}\Diamond z_{\langle LC(\mathsf{f})\rangle}$
7.	$\mathbf{true} \Rightarrow \mathbf{A}\bigcirc(\neg z \vee z_1)$		

We apply step resolution rules between 1 and 2, and 1 and 3. No more SRES rules are applicable. Formula 12 is an eventuality clause, and therefore, we are looking for a loop in $\neg p$ (see [13] for the formulation of the loop searching procedure). The desired loop, $\mathbf{E}\Box\mathbf{E}\bigcirc\neg p_{\langle LC(\mathsf{f})\rangle}$ (given that condition z_1 is satisfied) can be found considering clauses 10 and 11. Thus, we apply the TRES 3 rule to resolve this loop and clause 12, obtaining 16. Next we remove $\mathbf{E}\mathcal{W}$ from 16 deriving a purely classical Formula 17 (y is a new variable). Simplify the latter, apply TEMP (the 'temporising' rule, see [8], obtaining, in particular, 19 and 20, and then a series of SRES rules to newly generated clauses. Now, as no more SRES rules are applicable, we find another eventuality, Formula 13, and thus we next look for a loop in $\neg z$. This loop can be found considering Formulae 9 and 26: $\mathbf{A}\bigcirc\mathbf{A}\Box\neg z$ given that condition x_1 is satisfied. Thus, we can apply TRES 2 to resolve this loop and 13 deriving 27. Then we remove $\mathbf{E}\mathcal{W}$ from the latter (on step 28, where w is a new variable, we use only one of its conclusions). Applying simplification and temporising to 28 we obtain 29. The desired terminating clause $\mathbf{start} \Rightarrow \mathbf{false}$ is deduced by applying SRES 1 to steps 1, 15 and 23.

14.	$\mathbf{start}$	$\Rightarrow$	y	1, 2, *SRES* 1
15.	$\mathbf{start}$	$\Rightarrow$	x_1	1, 3, *SRES* 1
16.	y	$\Rightarrow$	$\mathbf{A}(\neg z_1 \mathcal{W} p)$	10, 11, 12 *TRES 3*
17.	y	$\Rightarrow$	$p \vee \neg z_1 \wedge v$	16, $\mathbf{A}\mathcal{W}$ Removal
18.	v	$\Rightarrow$	$\mathbf{A}\bigcirc(p \vee \neg z_1 \wedge v)$	16, $\mathbf{A}\mathcal{W}$ Removal
19.	$\mathbf{start}$	$\Rightarrow$	$\neg y \vee p \vee \neg z_1$	17, SIMP, TEMP
20.	$\mathbf{true}$	$\Rightarrow$	$\mathbf{A}\bigcirc(\neg y \vee p \vee \neg z_1)$	17, SIMP, TEMP
21.	$\mathbf{start}$	$\Rightarrow$	$p \vee \neg z_1$	14, 19, *SRES* 1
22.	$\mathbf{start}$	$\Rightarrow$	$p \vee \neg z$	5, 21, *SRES* 1
23.	$\mathbf{start}$	$\Rightarrow$	$\neg z$	4, 22, *SRES* 1
24.	x_1	$\Rightarrow$	$\mathbf{A}\bigcirc(p \vee \neg z_1)$	8, 20, *SRES* 3
25.	x_1	$\Rightarrow$	$\mathbf{A}\bigcirc(p \vee \neg z)$	7, 24, *SRES* 3
26.	x_1	$\Rightarrow$	$\mathbf{A}\bigcirc\neg z$	6, 25, *SRES* 3
27.	x	$\Rightarrow$	$\mathbf{E}(\neg x_1 \mathcal{W} z)_{\langle LC(\mathsf{f})\rangle}$	9, 26, 13 *TRES* 2
28.	x	$\Rightarrow$	$z \vee \neg x_1 \wedge w$	27 $\mathbf{E}\mathcal{W}$ Removal
29.	$\mathbf{start}$	$\Rightarrow$	$\neg x \vee z \vee \neg x_1$	28 SIMP, TEMP
30.	$\mathbf{start}$	$\Rightarrow$	$\mathbf{false}$	1, 15, 23 *SRES* 1

We have found a contradiction, meaning that $\mathrm{SNF}_{\mathrm{CTL}}(\ddagger)$, hence $\ddagger$ itself is unsatisfiable.

V. Conclusions and Future Work

In this paper we have introduced a formal framework for the deductive verification of modular specification. As a specification tool we use the branching-time temporal logic. Specified properties and requirements of the system are then translated into the language of a normal form, $\mathrm{SNF}_{\mathrm{CTL}}$, thus enabling the application of a powerful resolution method.

Future extensions of this work will be in the application of the Inferential Erotetic Logic (IEL) tools aiming at optimisation of the process of reconfiguration of a component model [15]. The Inferential Erotetic Logic (IEL) [31], [34] is a powerful tool in the area of analyzing and modelling such components of intelligent activity as planning, problem solving, and searching for information in massive data/knowledge bases [32]. Important developments within the framework of IEL are Erotetic Search Scenarios (ESS) [33] and Socratic Proofs (SP) [35]. ESS is based on the idea of providing conditional instructions for solving an initial problem, informing us which questions should be asked and when they should be asked. Moreover, an erotetic search scenario shows where to go if a direct answer to a query appears to be acceptable and does so with respect to any direct answer to each query. SP is a very specific technique, which reduces the complexity of standard problem-solving methods by using pure questioning only.

References

[1] R. Allen and D. Garlan A formal basis for architectural connection. In ACM Trans. Softw. Eng. Methodol., Vol 6, num 3, pages 213–249. ACM Press, NY, 1997.

[2] M. Astley and G. A. Agha. Customization and composition of distributed objects: Middleware abstractions for policy management. Proceedings of the ACM SIGSOFT 6th International Symposium on Foundations of Software Engineering(FSE). Pages 1–9, 1998.

[3] L. Bachmair and H. Ganzinger. A Theory of Resolution. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 2, Elsevier, 2001.

[4] T. Barros and L. Henrio and E. Madelaine. Behavioural Models for Hierarchical Components. In Proceedings of SPIN'05. Springer Verlag, 2005.

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

[6] F. Baude, D. Caromel, and M. Morel. From Distributed Objects to Hierarchical Grid Components. In *International Symposium on Distributed Objects and Applications (DOA)*, LNCS, Springer Verlag, 2003.

[7] P. Bidinger and J. Stefani. The kell calculus: operational semantics and type system. In *Proc. 6th IFIP FMOODS 03 Conference*, November 2003.

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

[9] 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.

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

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

[12] 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.

[13] A. Bolotov and C. Dixon. Resolution for Branching Time Temporal Logics: Applying the Temporal Resolution Rule. In *Proceedings of the 7th International Conference on Temporal Representation and Reasoning (TIME2000)*, pages 163–172, IEEE Computer Society.

[14] 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.

[15] A. Bolotov and P. Lupkowski and M. Urbanski Search and check. Problem solving by problem reduction. To be published in Proceedings of The 8th International Conference on Artificial Intelligence and Soft Computing, Zakopane, Poland, June 2006.

[16] J. Bradfield. and C. Stirling. Modal logics and mu-calculi. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier, North-Holland, 2001.

[17] E. Bruneton, T. Coupaye, and J. Stefani. Recursive and Dynamic Software Composition with Sharing. In *Proc. of the 7th Int. Workshop on Component-Oriented Programming (WCOP02)*, 2002.

[18] E. Bruneton, T. Coupaye, M. Leclercq, V. Qua, J.-B. Stefani. An Open Component Model and Its Support in Java. CBSE 2004.

[19] 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.

[20] E. A. Emerson. Automated reasoning about reactive systems. In *Logics for Concurrency: Structures Versus Automata, Proc. of International Workshop*, volume 1043 of Lecture Notes in Computer Science, pages 41–101. Springer, 1996.

[21] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *JCSS 30(1)*, pages 1–24, 1985.

[22] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not never" revisited: On branching versus linear time temporal logic. *JACM*, 33(1):151–178, 1986.

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

[24] M. Fisher. A Resolution Method for Temporal Logic. In *Proc. of the XII International Joint Conference on Artificial Intelligence (IJCAI)*, pages 99–104, 1991.

[25] M. Fisher and C. Dixon and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic (TOCL)*, 1(2):12–56, 2001.

[26] 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.

[27] T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Automata, Languages and Programming, Proc. 14 ICALP*, volume 267 of *Lecture Notes in Computer Science*, 269-279, 1987.

[28] 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.

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

[30] 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.

[31] A. Wisniewski. The Posing of Questions: Logical Foundations of Erotetic Inferences. Kluwer AP, 1995

[32] A. Wisniewski. Erotetic Logic and Explanation by Abnormic Hypoteses. Synthese 120, No. 3, pp. 295-309, 1999.

[33] A. Wisniewski. Erotetic search scenarios. Synthese 134, No 3, pp. 389-427, 2003

[34] A. Wisniewski. Questions and Inferences. Logique et Analyse, 173-174-175, pp. 5-43, 2001.

[35] A. Wisniewski. Socratic Proofs. Journal of Philosophical Logic 33, pp. 299-326, 2004.

[36] P. Wolper. On the relation of programs and computations to models of temporal logic. In L. Bolc and A. Szałas, editors, *Time and Logic, a computational approach*, chapter 3, pages 131–178. UCL Press Limited, 1995.