Ensuring Consistency of Dynamic Reconfiguration of Component-Based Systems

Hamza Zerguine  
MOVEP laboratory  
USTHB University  
Algiers, Algeria,  
Email: h.zerguine@usthb.dz

Nabila Salmi  
MOVEP Laboratory, USTHB  
Algiers, Algeria,  
LISTIC Laboratory, Université de Savoie  
Annecy le Vieux, France  
Email: nsalmi@usthb.dz

Malika Ioualalen  
MOVEP laboratory  
USTHB university  
Algiers, Algeria  
Email: m.ioualalen@usthb.dz

Abstract—The introduction of dynamic reconfiguration properties in a system can affect its performance and quality of service offered to users. Thus, performance prediction of component-based systems after reconfiguration is important to help software engineers to analyze their applications at the moment of reconfiguration and take a decision to keep or discard the analyzed reconfiguration, so that performance problems are avoided. In this case, the verification of functional and non-functional properties before and after reconfiguration becomes a challenge. In particular, when applying a reconfiguration on a system, the consistency of the new resulting architecture should be checked. To this aim, we describe, in this paper, a generic reconfiguration analysis approach which allows to check the reconfiguration consistency of a component-based architecture, starting from the architectural description of a component-based system. A case study of a system reconfiguration illustrates the effectiveness of our approach.

Keywords—Component-Based Systems; dynamic reconfiguration; formalization; consistency.

I. INTRODUCTION

Component-based approaches [1] are more and more essential for the development of systems and applications, to meet the challenges of engineering systems such as administration, autonomy. In this paradigm, components are developed in isolation or reused and are then assembled to build a Component Based System (CBS). Their objective is to enable a high degree of reusability of the software, rapid development (reducing the cost in terms of development time) and high quality since development is based on precompiled components. In this direction, numerous component models have been proposed (e.g., Enterprise Java Beans (EJB) [2], Corba Component Model (CCM) [3], Fractal [4], etc.). They operate different life-cycle stages, target different technical domains (embedded systems, distributed systems, etc.) and offer different degrees of tool support (textual modeling, graphical modeling, automated performance simulation, etc.).

Nowadays, systems need more and more to adapt their behaviour to their environment changes. To do that, they should dynamically add, remove or recompose components by the use of computational reflection. These abilities are called dynamic or runtime reconfiguration and constitute a key element to enable the adaptation of complex systems, such as embedded systems (mobile phones, PDAs, etc.) and service-oriented systems, to a changing environment. Moreover, dynamic system reconfiguration allows to achieve continuous availability of systems.

Dynamic reconfiguration techniques are promising solutions for building highly adaptable component-based systems. However, the introduction of dynamic reconfiguration properties in a system can affect its performance and quality of service offered to users. To avoid this, the design and verification of functional and non-functional properties of a reconfigured system become a challenge.

In this context, our long-term goal is to develop a methodology which allows analysis of component-based applications and their correction after reconfiguration, to help the decision to keep or discard the analyzed reconfiguration. The first property to ensure during analysis of such systems is consistency, which is defined as remaining compliant with their specification [5]. In this paper, we introduce a new formalism for checking consistency of dynamic reconfigurations of component-based systems. We provide this formalism for general component systems characterized by the most common component properties.

Outline. The structure of the paper is as follows. We discuss in Section II the related work. Then, we present in Section III the most important concepts of component-based systems. We detail our approach in Section IV and illustrate it in Section V. We conclude in Section VI and give future works.

II. RELATED WORK

Several approaches were proposed, during last years, for analysis of CBS; a few of them addressed dynamic reconfiguration.

In this context, two main proposals were given for dynamic reconfiguration analysis of CBS. First, Grassi et al. [6] proposed a metamodel called KLAPER, which includes a kernel modelling language. The main goal of this language is to act as a bridge between design models of component-based systems (built using heterogenous languages like Unified Modeling Language (UML) [7], Ontology Web Language (OWL) (OWL-S) [8], etc.) and performance analysis models (Markov chains [9], queueing networks [10], etc.). This first work did not address reconfiguration cases study. Later, in [11], an extension of KLAPER, called D-KLAPER, was given to support the model-based analysis of reconfigurable component-based systems, with a focus on the assessment of particular non-functional properties, namely performance and reliability.
The second work, defined by Leger [5], targeted dynamic reconfigurations reliability analysis for component-based systems, where an analysis approach for the Fractal component model was defined. The approach is summarized in three steps: the first step is a Fractal configuration modeling [5] step of a component-based configuration architecture; then, definition of mechanisms used for maintaining systems consistency during dynamic reconfigurations; finally, implementation of these mechanisms for checking reliable reconfiguration.

Besides, some other approaches were proposed for CBS for checking, in particular, consistency of CBS during dynamic reconfiguration. Warren et al. [12] proposed to do automatic runtime checks of reconfigurable component-based systems for the OpenRec framework [13]. A formal model based on ALLOY [14] was defined for that purpose. It allows architecture constraints expression and checking. Another work [15] has introduced an extension of the Fractal model [16], called Safran to enable the development of adaptive applications. It consists of a dedicated programming language for adaptation policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or.detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or.detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or attach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detach policies to or from Fractal basic components. Finally, M. Safran policies, as well as a mechanism to dynamically attach or detac...
where:
- name: is the unique name of the component C;
- granul: refers to granularity which can be Composite or Primitive;
- state: is the current state of the component C, which can be Started or Stopped.

Definition 3. A component interface i is defined as:

\[ i = \prec \text{itfc}, \text{role}, \text{visib}, \text{card}, \text{contig}, \text{sign} \succ \]

where:
- \text{itfc}: is the unique identifier of the interface (being of the form: component-name.interface-name);
- \text{role}: can be Client / Server (in the case of a service invocation interface) or Sink / Source (in the case of an event based interface);
- \text{visib}: refers to the visibility of the interface, which can be Internal or External;
- \text{card}: refers to the cardinality of the interface, which is Singleton or Collection;
- \text{contig}: characterizes the interface contingency, which may be Optional or Mandatory;
- \text{sign}: returns the interface signature.

Definition 4. A component binding b is defined with:

\[ b = \prec \text{itfc-clt}, \text{itfc-srv} \succ \]

where:
- \text{itfc-clt}: refers to the invoking interface, and can be Client or Sink;
- \text{itfc-srv}: refers to the service interface, and may be Server or Source.

2) Reconfiguration:

Definition 5. Let be a configuration \(C_g_1\) of a system \(S\). We define a reconfiguration \(R\) of \(S\), being in the configuration \(C_g_1\), as an ordered set of primitive operations applied on \(C_g_1\):

\[ R = \text{op}_1, \text{op}_2, \ldots, \text{op}_n, n \geq 1 \]

where \(\text{op}_i, i = 1..n, \) is one of the following reconfiguration operations:

1) Delete a component
2) Add a component
3) Replace a component
4) Delete a binding
5) Add a binding

The resulted configuration after application of \(R\) is denoted \(C_g_2\).

We denote this by:

\[ C_g_1 \xrightarrow{R} C_g_2 \]

3) Predefined functions: To be able to specify constraints required for performing properly a reconfiguration, we need a set of predefined functions. For this objective, we propose the following functions:

1) \(\text{CFather}(cp)\): returns the parent of the component \(cp\);
2) \(\text{CInterfaces}(cp)\): returns the interfaces list of the component \(cp\);
3) \(\text{CType}(cp)\): returns the type of the component \(cp\);
4) \(\text{IComponent}(i)\): returns the owner of the interface \(i\);
5) \(\text{IType}(i)\): returns the type of the interface \(i\).

B. Constraints

To ensure the correction of a reconfiguration \(R\) applied on a system \(S\), we define two sets of constraints:

- Constraints on primitive reconfiguration operations:
  Should be checked after each primitive operation.
- Global constraints: should be checked after the whole reconfiguration.

In the following, we specify these two sets of constraints.

1) Constraints on primitive reconfiguration operations:

Let \(op\) be a primitive reconfiguration operation, applied on a configuration \(C_g_1\) of a system \(S\), resulting in the configuration \(C_g_2\), where:

\[ C_g_1 = \prec C_1, I_1, B_1 \succ \]
\[ C_g_2 = \prec C_2, I_2, B_2 \succ \]

We denote this by:

\[ C_g_1 \xrightarrow{op} C_g_2 \]

In the following, we consider:

- A component \(cp = \prec \text{name}, \text{granul}, \text{statut} \succ \)
- A binding \(b = \prec \text{iclt}, \text{isrv} \succ \)

Primitive reconfiguration operations, applied on components \(cp\) and \(cp'\), are denoted as follows:

1) Delete a component \(cp\):

\[ \text{del}\_\text{comp}(cp) \]

2) Add a component \(cp\):

\[ \text{add}\_\text{comp}(cp) \]

3) Replace a component \(cp\) by another \(cp'\):

\[ \text{Repl}\_\text{comp}(cp, cp') \]

4) Delete a binding \(b\):

\[ \text{del}\_\text{bdg}(b) \]

5) Add a binding \(b\):

\[ \text{add}\_\text{bdg}(b) \]

Table I gives the required constraints to be satisfied after each reconfiguration operation.
2) Global constraints: Let $R$ be a reconfiguration that will be applied to a configuration $Cg_1$ of a system $S$, giving as a result a configuration $Cg_2$:

$$Cg_1 \xrightarrow{R} Cg_2$$

with: $R = op_1, op_2, ..., op_n, n \geq 1$

We specify the following constraints, which must be satisfied by $Cg_2$:

1) $\forall b \in B_2, b.iclt.role = Client \land b.isrv.role = Server / Source$
2) $\forall b \in B_2, (b.iclt.contig = Mandatory) \Rightarrow (b.isrv.contig = Mandatory)$
3) $\forall b, b' \in B_2, b.iclt \neq b'.iclt$
4) $\forall b \in B_2, (CFather(IComponent(b.iclt)) = CFather(IComponent(b.isrv))) \lor (b.iclt.visib = Internal \land IComponent(b.iclt) = CFather(IComponent(b.isrv)))$
5) $\forall i \in I_2 (i.role = Client \land i.contig = Mandatory) \Rightarrow \exists! b \in B_2 (b.iclt = 1)$
6) $\forall b \in B_2, IType(b.isrv) \subseteq IType(b.iclt)$

C. Consistency of a configuration

Theorem 1. A reconfiguration $R$, applied to a configuration $Cg_1$ of a system $S$, is valid if the resulting configuration $Cg_2$ satisfies all constraints defined on primitive reconfiguration operations and global constraints.

Theorem 2. A configuration $Cg_1$ of a system $S$ is consistent after a reconfiguration $R$ if $R$ is valid.

V. ILLUSTRATION

To illustrate our approach, we use a navigator application similar to Mozilla already used in [20]. In such applications, components are usually equipped with an install manifest in XML format, allowing, among other things, to deliver the information needed to manage the version compatibility between components.

![Initial configuration](image)

So, the architecture of the application consists of a composite component $MAIN$ composed of three primitive components (Figure 1):

1) $M$, the main application (e.g., Firefox);
2) $E$, an already installed plugin;
3) $VM$, a version manager component.

Each of the components $M$ and $E$ have an interface $h$ with a signature $H$ being respectively a client and server interface. They also each have a server interface $im$ of signature $InstallM$. $M$ has an additional server interface $g$ of signature $G$, being the main interface exported to the global external interface of the application.

The Main composite exports business methods from $M$ and supplies update, a control method implementing the upgrade operation. This method looks for a component with same id as $E$, having a more recent version and being compatible with $M$. In case of success, it replaces $E$ with the new component.

Based on our formalization, we specify the initial configuration of Figure 1 as follows:

$$Cg_1 = \{ C_1, I_1, B_1 \}$$
which removes the plugin E follows:

consistent configuration (given in Figure 2), which is defined as reconfiguration where:

\[ C_1 = (\text{Main}, M, VM, E) \]
\[ I_1 = (\text{Main}.g, M.g, M.im, VM.a1, VM.a2, E.im, E.h) \]
\[ B_1 = (b_1, b_2, b_3, b_4) \]

where:

\[ M = (M, \text{Primitive}, \text{Started}) \]
\[ VM = (VM, \text{Primitive}, \text{Started}) \]
\[ E = (E, \text{Primitive}, \text{Started}) \]
\[ \text{Main.g} = (\text{Main.g}, \text{Server}, \text{External}, \text{Singleton}, \text{Optional}, G) \]
\[ M.g = (M.g, \text{Server}, \text{External}, \text{Singleton}, \text{Optional}, G) \]
\[ M.im = (M.im, \text{Server}, \text{External}, \text{Singleton}, \text{Mandatory}, \text{InstallMF}) \]
\[ VM.a1 = (VM.a1, \text{Client}, \text{External}, \text{Singleton}, \text{optional}, \text{InstallMF}) \]
\[ VM.a2 = (VM.a2, \text{Client}, \text{External}, \text{Singleton}, \text{optional}, \text{InstallMF}) \]
\[ E.im = (E.im, \text{Server}, \text{External}, \text{Singleton}, \text{Mandatory}, \text{InstallMF}) \]
\[ b_1 = (\text{Main.g}, M.g) \]
\[ b_2 = (VM.a1, M.im) \]
\[ b_3 = (VM.a2, E.im) \]
\[ b_4 = (M.h, E.h) \]

When applying on this configuration a reconfiguration R, which removes the plugin E, we model this by the following reconfiguration R:

\[ R = op_1, op_2, op_3 \]

where:

\[ op_1 : \text{Del}\_\text{comp}(E), \]
\[ op_2 : \text{Del}\_\text{bdg}(b_3), \]
\[ op_3 : \text{Del}\_\text{bdg}(b_4). \]

This resulted configuration is valid because it provides a new consistent configuration (given in Figure 2), which is defined as follows:

\[ Cg_2 = \langle C_2, I_2, B_2 \rangle \]

where:

\[ C_2 = (\text{Main}, M, VM) \]
\[ I_2 = (\text{Main.g}, M.g, M.im, M.h, VM.a1, VM.a2) \]
\[ B_2 = (b_1) \]

VI. CONCLUSION

In this paper, we presented a new formalism for checking consistency of dynamic reconfigurations of general component-based systems. For this purpose, we introduced formal concepts for modelling a component-based configuration and reconfiguration operations. We also defined required constraints that must be satisfied by the new configuration resulting after applying reconfiguration, to ensure consistency of the system.

Our approach can be instanciated to any existing component model, allowing thus genericity of the formalism.

Work is in progress to achieve automation of the proposed approach, by providing a toolbox based on the FOCALIZE programming environment [21]. This latter is based on a functional programming language with object-oriented features and allows to write formal specifications and proofs of designed programs. Proofs are build using the automated theorem prover Zenon [22] and Coq proof-assistant [23]. Future work also include modeling CBS before and after reconfiguration to allow quantitative analysis of CBS.

REFERENCES