Memory Resource Estimation of Component-Based Systems

Trinh Dong Nguyen(✉)

HPU, Haiphong, Vietnam
dongnt@hpu.edu.vn

Abstract. Traditional relational interface theory focuses on expressing functional aspects of software components. We extend the theory by adding resource specification to reason for the quality of composite components in terms of resource efficiency. For practical application, we instantiate interface using automata and present algorithms to check if a component system met the predefined resource requirements. In particular, we can answer if a component can be plugged into an environment of whether it is a refinement of another component.

Keywords: Estimation of memory resources · Resource estimation component-based systems · Model compositions · Interface based designs · Interface compositions

1 Introduction

Estimating resources of component-based systems is one of the important issues in software engineering. The estimation covers many features in a system such as memory resources, time resources, and the others in which the estimation of memory is addressed. The memory resource identification of component-based systems aims for forecasting memory resource of a system that consumes in operating. It then calculates supreme memory resources of a system, whether they satisfy the system's requirement. In particular, those can be fulfilled in design phase. This is important in utilization of resources in the embedded system whose the memory optimization is always considered. Hence, how do estimate memory resources of a component-based system in design phase? The problem will be solved in the next sections.

The first, the paper extends the relational interface [6] and timed design in [1] for memory constraints. Components in a system are described by relational interfaces with memory design constraints. Hence, they are composed together depending on plugable, refinement, parallel and sequential operations to construct complex systems. The second, the paper calculates the memory resource which is used in an interface, and predicts memory resources for pluggable, refinement, parallel and sequential compositions. The third, the paper proposes algorithms to estimate the memory resources for above items within a system. By using this theory, the estimation of memory resources of a component-based system can be implemented effectively. This estimation can be done at the earliest stage of a system development.

In general, the aim of this paper uses the memory resource design pattern with memory resource constraints for specifying component-based systems and estimating memory resources. Especially, the proposed algorithms in this paper can estimate memory resource of a component-based system in design phase.

The paper is organized as follows: The next section is related works. Section 3 describes the specification of components by interfaces with their environment, and modeling them by finite automata. Simultaneously, this section also introduces to the pluggable, refinement, parallel and sequential composition of those automata. Section 4 estimates memory resources for automaton interface, pluggability of automata interfaces, composition of automata interfaces, and proposes algorithms to compute those resources. The last section is the conclusion of our paper.

2 Related Work

Interface theory is one of promise approaches for component-based system in which relational interface emerges as most efficiency method [6]. However, this approach has just captured the relation between input and output sets of components in terms of first-order logic, and other properties have not addressed such as memory resources property, timed property, etc. D.V. Hung et al. use the UTP notation to denote timed design pattern in [1]. Those signatures are concise, easy to depict components in a system. However, the authors use for specifying time feature purpose.

In the memory resource estimation, recently, A.V. Fioukov et al. introduced a method to calculate the static properties of an architecture depending on a framework in which estimating memory resource was one of aspects, and this work based on the source codes [3]. The authors used two approaches bottom-up and top-down algorithm to predict memory size and to evaluate static properties of components, the contribution has only applied to *Koala Component Model* [2,3]. Johan Muskens and Michel Chaudron proposed an approach to predict resources of a system in run-time based on scenario. However, this method cannot apply for interface based design [5]. Merijn de Jonge et al. proposed a method to estimate the resource of a system in run-time. This proposal focus on evaluation of memory resource relying on modeling behaviors of a component by sequence of messages, and called scenarios [4].

In general, all above works consider various aspects of systems and specific applications such as *Koala Component Model*, *Robocop*, but they do not formalize a general theory for component-based systems. Furthermore, memory resources cannot predict in early state of the software development.

3 Interface and Interface Modeling

3.1 Memory Resource Design

A component supplies services to its environment, it invades memory resources within a system. Therefore, a component consumes memory resource depending on the variables size and what kind of services for which it provides. In this part, the paper uses a notion *memory resource design*, denoted μ. Let X be a set of input ports, Y be a set of output ports, C be upper bound memory resources of an interface that stores the memory size value of an interface at a specific run-time.

Definition 1 (Memory Resource Design). *A memory resource design is of the form $\mu = p \vdash (R, C)$, where p is a guard over input set X, called precondition, R is a first-order logic formula, depicts the relation between X and Y and called post-condition, and C is an upper bound memory resource.*

This Definition depicts that a *memory resource design* as an atomic constraint in an interface. For an assignment $\mathcal{V}$ over X, the values satisfy precondition p will activate the interface and give values that also satisfy post-condition R at output ports Y. In the software evolutionary context, a *memory resource design* can be replaced by the other provided that the new one supplies better services and using fewer resources than the original one. The refinement of two memory resource designs is defined as follows:

Definition 2 (Memory Design Refinement). *Given two memory resource designs $\mu = p \vdash (R, C)$ and $\mu' = p' \vdash (R', C')$, μ is said to be a refinement of μ', denoted as $\mu \sqsubseteq \mu'$ iff $p' \Rightarrow p$, $R \Rightarrow R'$, and $C \leq C'$. When $\mu \sqsubseteq \mu'$ and $\mu' \sqsubseteq \mu$ we say μ and μ' are equivalent.*

Let $\mathbb{N}$ be the set of natural numbers, an assignment over $X \cup Y$ is a pair $(\mathcal{V}, t)$, denoted as γ and called a computation step, where $\mathcal{V}$ is an assignment over variables in $X \cup Y$, and t is a memory capacity using for the $\mathcal{V}$, $t \in \mathbb{N}$. The γ is a computation step of an interface iff γ satisfies a memory resource design $\mu = p \vdash (R, C)$ in that interface, signified $(\mathcal{V}, t) \models \mu$, iff $\mathcal{V}|_X \models p$, $\mathcal{V} \models R$ and $sizeof(t) \leq C$, where $\mathcal{V}|_X$ is an assignment on variables X, and $sizeof(t)$ is the capacity of memory. If for all $p \equiv false$, no computation step γ can satisfy μ. Considering two pairs $(\mathcal{V}, t)$ and $(\mathcal{V}', t')$, the $(\mathcal{V}, t)$ is equal to $(\mathcal{V}', t')$ iff $\mathcal{V} = \mathcal{V}'$ and $sizeof(t) = sizeof(t')$. Given any equivalent γ, γ' and a memory resource design μ, this only holds that $\gamma \models \mu$ if and only if $\gamma' \models \mu$. A computation step $\gamma = (\mathcal{V}, t)$ is said to be before a computation step $\gamma' = (\mathcal{V}', t')$, in other words, γ' is after γ. Given a sequence of consecutive computation step $s = (\gamma_1 \ldots \gamma_n)$, where $n \in \mathbb{N}$, for all $i \in n$ such that γ_{i+1} is after γ_i to be called a state.

Let $\mathcal{S}(X, Y)$ denote the set of all states, $\mathcal{M}(X, Y)$ denote the set of all *memory resource designs* over the set of $(X \cup Y)$. A relational interface with memory constraint is defined as follows:

Definition 3 (Interface). *A relational interface with memory resources is a triple $\mathcal{I} = \langle X, Y, \xi \rangle$, $X \cap Y = \varnothing$, $\xi : \mathcal{S}(X, Y) \nrightarrow \mathcal{M}(X, Y)$ satisfying the formula $\xi(s) = \mu$. If $s = \epsilon$ implies $\xi(s) = \mu_0$, where ϵ is an initial state, and μ_0 is a memory resource design corresponding to initial state. In the contrary, let s be a sequence of $(\gamma_1 \gamma_2 \ldots \gamma_n)$, if $\xi(\gamma_1 \gamma_2 \ldots \gamma_n)$ is defined, then $\xi(\gamma_1 \gamma_2 \ldots \gamma_{n-1}) = \mu_{n-1}$ is also defined, and $\gamma_n \models \mu_{n-1}$. When $\xi(s)$ is defined, s is said a reachable state of $\mathcal{I}$. Let $\mathcal{R}(\mathcal{I})$ denote the set of all reachable states of interface $\mathcal{I}$.*

Example 1. This example illustrates a relational interface.

$$\mathcal{I} = \langle \{x\}, \{y\}, \{(x \geq 80 \wedge x \leq 260) \vdash (x \geq 80 \wedge x \leq 260 \wedge y = 220, 9)\}\rangle$$

This interface describes a component that has only one input port $\{x\}$, one output port $\{y\}$ and ξ guarantees that if $x \geq 80 \wedge x \leq 260$ then y always is equal to 220. The number 9 indicates that the interface consumes at most 9 memory units.

An interface is activated if it is plugged to its environments. Those environments supply resources to interfaces such as data, memory resources, etc. No all behaviors of an interface is implemented, but only behaviors satisfy conditions of the environment. Suppose a sequence of behaviors $(\mathcal{V}_1, t_1)(\mathcal{V}_2, t_2)\ldots(\mathcal{V}_m, t_m)$, where $\mathcal{V}_i$ is an assignment over $(X \cup Y)$, $t_i \in \mathbb{N}$, t_i is a current memory capacity at computation step i^{th}, $t_1 = 0$. An environment gives an assignment over input variables X of an interface by $\mathcal{V}_i|_X$ and expects values at output variables $\mathcal{V}_i|_Y$. Therefore, the assignment $\mathcal{V}_i$ consumes a memory capacity that is stored in variable t, where $sizeof(t_i) \leq C_i$. Let $\mathcal{P}(X,Y)$ be the set of all computation step sequences on (X,Y) of an environment.

Definition 4 (Environment). *An environment is a triple $E = \langle X, Y, \delta\rangle$, where $\delta : \mathcal{P}(X,Y) \nrightarrow \mathcal{M}(X,Y)$. E is defined as $\mathcal{I}$ excepts the sequence $w = (w_1 w_2 \ldots w_n)$ to be an interaction of E with $\mathcal{I}$. The w is said to be a reachable state of environment E. Let $\Pi(E)$ denote the set of all reachable states of environment E.*

3.2 Interface and Environment Modeling

The behaviors of an interface are infinite, therefore an interface $\mathcal{I} = \langle X, Y, \xi\rangle$, where ξ is a partial function from infinite set $\mathcal{S}(X,Y)$ to the set $\mathcal{M}(X,Y)$ need to be finitely represented. This part describes a method to represent interfaces and environments by label automata.

Definition 5 (Labeled Automata). *A labeled automaton M is a tuple $M = \langle Q, X, Y, q_0, T, l_s, l_t\rangle$, where Q is a finite set of locations, X and Y are sets of input and output ports, respectively, and $X \cap Y = \varnothing$, $q_0 \in Q$ is an initial state of M, $T \subseteq Q \times Q$ is a set of transitions, l_s and l_t are labeling functions. The labeling functions $l_s : Q \rightarrow \mathcal{M}(X,Y)$ associates each location in M with a memory resource design, and $l_t : T \rightarrow \mathcal{F}(X \cup Y)$ associates each transition in T with a guard formula. For any two different transitions (q, q') and (q, q''), the formula $l_t(q, q'') \wedge l_t(q, q') \Rightarrow false$ in order to make M deterministic.*

Hence, how to use a labeled automaton presents the behaviors of an interface. Let $\mathcal{A}(X \cup Y)$ be the set of all computation steps over $(X \cup Y)$. Suppose an assignment $\mathcal{V}_i$ in one of any sequences belonging to $\mathcal{A}(X \cup Y)$, i.e., $\mathcal{V}_i \in \mathcal{A}(X \cup Y)$, $i \leq n$. The assignment $\mathcal{V}_i$ inputs a set of values to input ports X and expects a set of values at output ports Y. This assignment has to satisfy one of memory resource designs that are available in an interface. Let $f : \mathcal{A}(X \cup Y)^* \nrightarrow \mathcal{M}(X \cup Y)$, i.e., M represents the partial function f. The labeled automaton M depicts partial function f as follows: For the initial state ϵ, $f(\epsilon) = l_s(q_0)$, i.e., ϵ leads M to q_0, and for any sequence $s \in \mathcal{A}(X \cup Y)^*$, if $f(s) = p \vdash (R, C)$ then s leads M to location q. According to the Definition 5, at current time there is at most one location q' such that $\mathcal{V} \models l_t(q, q')$. Therefore, an automaton describes an interface is defined as follows:

Definition 6 (Automata Interface). *A labeled automaton M is a description of an interface $\mathcal{I}$, and the interface $\mathcal{I}$ becomes an automaton interface iff for any sequence $(\mathcal{V}_1, t_1), \ldots, (\mathcal{V}_k, t_k)$ in labeled automaton M such that ($k \geq 0$, and the case $k = 0$ corresponds to the state ϵ), the value $\xi((\mathcal{V}_1, t_1), \ldots, (\mathcal{V}_k, t_k))$ is defined exactly when $f(\mathcal{V}_1, \ldots, \mathcal{V}_k)$ is defined and $\xi((\mathcal{V}_1, t_1), \ldots, (\mathcal{V}_k, t_1)) = f(\mathcal{V}_1, \ldots, \mathcal{V}_k)$ provided that $sizeof(t_i) \leq C_i$, where $f(\mathcal{V}_1, \ldots, \mathcal{V}_i) = p_i \vdash (R_i, C_i)$, $i \in n$.*

In order to use M describing an environment. The labeled automaton M gets over all sequence of behaviors of an environment such that satisfying all requirements of the environment. Given a sequence $(\mathcal{V}_1, t_1)(\mathcal{V}_2, t_2) \ldots (\mathcal{V}_n, t_n)$, for each $(\mathcal{V}_i, t_i)$, $i \leq n$, the function δ always is defined, i.e., labeling function l_s mounts a label corresponding to a location within automaton M. The $\delta((\mathcal{V}_1, t_1) \ldots (\mathcal{V}_k, t_k)) = p \vdash (R, C)$ iff there is a derivation $q_0 \xrightarrow{(\mathcal{V}_0, t_0)} \ldots \xrightarrow{(\mathcal{V}_{k-1}, t_{k-1})} q_k$ in automaton M such that $l_s(q_k) = p \vdash (R, C)$, and $\mathcal{V}_i \models p_{i-1} \wedge R_{i-1}$, where $l_s(q_{i-1}) = p_{i-1} \vdash (R_{i-1}, C_{i-1})$, $i = 1, \ldots, k-1$. All the derivations in automaton M are distinct sequences.

Definition 7 (Automata Environment). *A labeled automaton M is a description of Environment $E = \langle X, Y, \delta \rangle$ and environment E becomes an automata environment iff for any sequence $(\mathcal{V}_1, t_1), \ldots, (\mathcal{V}_k, t_k)$ in labeled automaton M such that ($k \geq 0$, and the case $k = 0$ corresponds to the state ϵ), the sequence $\delta((\mathcal{V}_1, t_1), \ldots, (\mathcal{V}_k, t_k))$ is defined exactly when $f(\mathcal{V}_i, \ldots, \mathcal{V}_k)$ is defined and $\delta((\mathcal{V}_1, t_1), \ldots, (\mathcal{V}_k, t_1)) = f(\mathcal{V}_1, \ldots, \mathcal{V}_k)$ provided that $sizeof(t_i) \leq C_i$, where $f(\mathcal{V}_1, \ldots, \mathcal{V}_i) = p_i \vdash (R_i, C_i)$, $i \in n$.*

3.3 Automata Interface Composition

This section considers three operations which are pluggable, parallel and sequential compositions. Given $\mathcal{I}, \mathcal{I}', E$ represented by $M = \langle Q, X, Y, q_0, T, l_s, l_t \rangle$, $M' = \langle Q', X', Y', q_0', T', l_s', l_t' \rangle$, and $M^e = \langle Q^e, X^e, Y^e, q_0^e, T^e, l_s^e, l_t^e \rangle$, resp. $\mathcal{I}$ and $\mathcal{I}'$ compose together in parallel, denoted $\mathcal{I} || \mathcal{I}'$. $\mathcal{I}$ and $\mathcal{I}'$ compose together in sequence, denoted $\mathcal{I}._\theta \mathcal{I}'$, and the interface $\mathcal{I}$ plugs to E, denoted $\mathcal{I} \multimapboth E$. Given the automata interface $\mathcal{I}$ and the automata environment E, a sequence $(\mathcal{V}_1, t_1), (\mathcal{V}_2, t_2), \ldots, (\mathcal{V}_n, t_n)$ of environment E, and a set of memory resource designs $\mathcal{M}$ with $\mu \in \mathcal{M}$. For any an assignment $\mathcal{V}_i$, $i \in n$, the automaton E offers a set of values over X that satisfies p_{i-1} to the automaton $\mathcal{I}$, and expects results Y by $\mathcal{V}_i$ from the interface, and the outputs satisfies the post-condition R_{i-1}. i.e., $\mathcal{V}_i \models p_{i-1} \wedge R_{i-1}$. When E invokes $\mathcal{I}$, a protocol is created between

E and $\mathcal{I}$, this means that the specification of automaton interface $\mathcal{I}$ satisfies the requirement of automaton environment E at anytime in the process of interaction. Let μ be a memory resource design of $\mathcal{I}$, and μ^e be a memory resource design of E. For any computation step $(\mathcal{V}_i, t_i)$ is assigned from the E to the $\mathcal{I}$, the $(\mathcal{V}_i, t_i) \models \mu^e_{i-1} \wedge \mu_{i-1}$, i.e., the formulas $p^e_{i-1} \Rightarrow p_{i-1}$ and $p^e_{i-1} \wedge R_{i-1} \Rightarrow R^e_{i-1}$ hold. The memory resource uses for a computation step of the environment and the interface to be calculated as follows: Let $C_{\multimapboth}$ be an upper bound memory resource of $\mathcal{I} \multimapboth E$, the supreme memory $C_{\multimapboth} = C + C^e$.

Definition 8 (Pluggability). *Given $\mathcal{I}$ is represented by M and E is represented by M^e. The $\mathcal{I}$ is pluggable to the E, denoted $\mathcal{I} \multimapboth E$, iff $X = X^e$, $Y = Y^e$ and the following conditions are satisfied:*

1. *Let $\delta(\epsilon) = p^e_0 \vdash (R^e_0, C^e_0)$ and $\xi(\epsilon) = p_0 \vdash (R_0, C_0)$, where ϵ is an initial state of both E and $\mathcal{I}$. Then, $p^e_0 \Rightarrow p_0$, $p^e_0 \wedge R_0 \Rightarrow R^e_0$. For any $\mathcal{V}_1$ such that $\mathcal{V}_1 \models p^e_0 \wedge R_0$, if $\delta(\mathcal{V}_1, t^e_1)$ is defined then $\xi(\mathcal{V}_1, t_1)$ is also defined, and the pair $(\mathcal{V}_1, t_1)$ is called reachable state of $\mathcal{I} \multimapboth E$. The memory constraints for state ϵ is defined as follows: The $C_{\multimapboth_0} = C^e_0 + C_0$.*
2. *Let $n \in \mathbb{N}, n \geq 1$ and let $w_n = (\mathcal{V}_1, t^e_1), \ldots, (\mathcal{V}_n, t^e_n)$ be an interaction behavior sequence of E with $\mathcal{I}$, and $s_n = (\mathcal{V}_1, t_1)...(\mathcal{V}_n, t_n)$ be a computation sequence of $\mathcal{I}$ interacts with E. Furthermore, let $\delta(w_n) = p^e_n \vdash (R^e_n, C^e_n)$ and $\xi(s_n) = p_n \vdash (R_n, C_n)$, then, $p^e_n \Rightarrow p_n$, $p^e_n \wedge R_n \Rightarrow R^e_n$. For any $\mathcal{V}_{n+1}$ such that $\mathcal{V}_{n+1} \models p^e_n \wedge R_n$, if $w_{n+1} = (\mathcal{V}_1, t^e_1), \ldots, (\mathcal{V}_n, t^e_n)(\mathcal{V}_{n+1}, t^e_{n+1})$ is a reachable state of E, then $s_{n+1} = (\mathcal{V}_1, t_1), \ldots, (\mathcal{V}_n, t_n)(\mathcal{V}_{n+1}, t_{n+1})$ is also a reachable state of $\mathcal{I}$, and s_{n+1} is called a reachable state of $\mathcal{I}$ w.r.t. w_{n+1} while w_{n+1} is called a behavior of E w.r.t. $\mathcal{I}$. The memory constraints for reachable state (w_{n+1}, s_{n+1}) is defined as follows: The upper bound memory is $C_{\multimapboth_{n+1}} = C^e_{n+1} + C_{n+1}$.*

For any pair $(\mathcal{V}_i, t^e_i)$ in a reachable behavior $w = (\mathcal{V}_1, t^e_1)(\mathcal{V}_2, t^e_2), \ldots, (\mathcal{V}_n, t^e_n)$ of E is expendable, then it makes the state $s = (\mathcal{V}_1, t_1)(\mathcal{V}_2, t_2), \ldots, (\mathcal{V}_n, t_n)$ of the interface $\mathcal{I}$ is also expendable.

Definition 9 (Parallel Composition). *Given two automata interfaces $\mathcal{I}, \mathcal{I}'$ represented by labeled automata M, M' respectively, such that $(X \cup Y) \cap (X' \cup Y') = \varnothing$. The parallel composition $\mathcal{I} || \mathcal{I}' = \langle X \cup X', Y \cup Y', \xi'' \rangle$, where $\xi'' : \mathcal{S}(X \cup X', Y \cup Y') \nrightarrow \mathcal{M}(X \cup X', Y \cup Y')$. Suppose $s = (\mathcal{V}_1, t''_1)...(\mathcal{V}_n, t''_n)$, $s \in \mathcal{S}(X \cup X', Y \cup Y')$, $\xi''(s)$ is defined as follows:*

- *$\xi((\mathcal{V}_1|_{X \cup Y}, t_1), \ldots, (\mathcal{V}_n|_{X \cup Y}, t_n)) = p \vdash (R, C)$, and*
- *$\xi'((\mathcal{V}_1|_{X' \cup Y'}, t'_1), \ldots, (\mathcal{V}_n|_{X' \cup Y'}, t'_n)) = p' \vdash (R', C')$,*

where $\mathcal{V}_i|_{X \cup Y}$ and $\mathcal{V}_i|_{X' \cup Y'}$ are the restriction of $\mathcal{V}_i$ over $X \cup Y$ and $X' \cup Y'$, $i \in n$, respectively. $\xi''(s) = p \wedge p' \vdash (R \wedge R', C + C')$ and t''_n satisfies that $sizeof(t''_n) \leq C + C'$.

For the sequential connection, given two automata interfaces $\mathcal{I}, \mathcal{I}'$ such that an input of the second connects to only one output of the first and $(X \cup Y) \cap (X' \cup Y') = \varnothing$. A connection from $\mathcal{I}$ to $\mathcal{I}'$ is a set of pairs $\theta \subseteq Y \times X'$ that satisfies $\forall (y, x), (y', x') \in \theta.(x = x' \Rightarrow y = y')$. Let $X_\theta = \{x \in X' | \exists y \in Y.(y, x) \in \theta\}$. An assignment $\mathcal{V}$ over $(X \cup X' \cup Y \cup Y')$ passes through a connection θ by an assignment $\mathcal{V}_\theta$ over $((X \cup X') \setminus X_\theta) \cup Y \cup Y'$ such that $\mathcal{V}_\theta|_{((X \cup X') \setminus X_\theta) \cup Y \cup Y'} = \mathcal{V}|_{((X \cup X') \setminus X_\theta) \cup Y \cup Y'}$, and for $x \in X_\theta$ then $\mathcal{V}_\theta(x) = \mathcal{V}(y)$, where y is the unique element in Y and $(y, x) \in \theta$. Therefore $l_\theta = \bigwedge_{(y,x) \in \theta} (x = y)$.

Definition 10 (Sequential Composition). *Let $\mathcal{I}, \mathcal{I}'$ be represented by M, M' respectively, such that $(X \cup Y) \cap (X' \cup Y') = \varnothing$. A sequential composition of $\mathcal{I}$ and $\mathcal{I}'$ w.r.t connection θ, denoted by $\mathcal{I}.{}_\theta\mathcal{I}'$ is an automaton interface $\mathcal{I}'' = \langle X'', Y'', \xi'' \rangle$, where $X'' = (X \cup X') \setminus X_\theta$, $Y'' = Y \cup Y'$. For $s = (\mathcal{V}_1, t''_1), \ldots, (\mathcal{V}_n, t''_n) \in \mathcal{S}(X'', Y'')$, $\xi''(s)$ is defined iff the both formulas $\xi((\mathcal{V}_{\theta 1}|_{X \cup Y}, t_1), \ldots, (\mathcal{V}_{\theta n}|_{X \cup Y}, t_n)) = p \vdash (R, C)$ and $\xi'((\mathcal{V}_{\theta 1}|_{X' \cup Y'}, t'_1), \ldots, (\mathcal{V}_{\theta n}|_{X' \cup Y'}, t'_n)) = p' \vdash (R', C')$ are defined, and then $\xi''(s) = p \wedge \exists Y.(R \wedge p' \wedge l_\theta) \vdash (R \wedge R' \wedge l_\theta \wedge p', max(C, C'))$ and t''_n satisfies that $sizeof(t''_n) \leq max(C, C')$.*

4 Estimating Memory Resources

In this section, we introduce a method to calculate the memory resources of an interface. Given a *memory resource design* $\mu = p \vdash (R, C)$, we can estimate the memory resource that is consumed when the interface was enabled. Therefore, memory resource in a given automaton interface $\mathcal{I}$ can be computed as follows: The supreme memory resource of an interface, denoted $U = max(C_i)$, $i \in n$.

Lemma 1. *The memory resource consumption of $\mathcal{I}$ has estimated based on the memory resource designs in $\mathcal{I}$.*

The algorithm is illustrated below computing the maximum lower bound and upper bound memory capacity of an interface.

Algorithm 1. The memory resource estimation for an automaton interface

```
Input: Automata interface $\mathcal{I}$
Output: The memory resource consumption of an automaton interface
1 begin
2     $MaxUpperbound \leftarrow 0.$
3     foreach $\mu \in \mathcal{M}$ do
4         $MaxUpperbound \leftarrow Max(MaxUpperbound, \mu.C)$
5     end
6     return $MaxUpperbound$
7 end
```

Lemma 2. *Given $\mathcal{I}$ and E, the memory resource consumption for pluggable operation has been estimated iff the $\mathcal{I}$ plugs to the E.*

Algorithm 2. Get memory resource consumption for $\mathcal{I}$ plugs to E.

```
Input: $M = \langle Q, X, Y, q_0, T, l_s, l_t \rangle$ and $M^e = \langle Q^e, X^e, Y^e, q_0^e, T^e, l_s^e, l_t^e \rangle$.
Output: Memory consumption of $\mathcal{I} \multimap E$
begin
    Let $f \subseteq Q^e \times Q$, $f \leftarrow \{(q_0^e, q_0)\}$ and $(q_0^e, q_0)$ be unmarked.
    while (true) do
        if GetAllMarked(f)= true then
            return true
        else
            $CurrentElement \leftarrow GetUnmarked((q^e, q) \in f))$
            let $l_s(q) = p \vdash (R, C)$ and $l'_s(q') = p' \vdash (R', C')$.
            $q^e_{next}List \leftarrow GetReachableE(Q^e)$ such that
            $F_{(q^e, q^e_{next})} : := p^e \wedge R \wedge l^e_t(q^e, q^e_{next})$ is satisfiable.
            $q_{next}List \leftarrow GetReachableI(q^e_{next}List)$ such that $(q, q_{next}) \in T$ and
            $l_t(q, q_{next}) \wedge F_{(q^e, q^e_{next})}$ is satisfiable.
            if $(F_{(q^e, q^e_{next})} \Rightarrow \bigvee_{q_{next} \in q_{next}List} l_t(q, q_{next})) = false$ then
                return false
            else
                $f \leftarrow (q^e_{next}, q_{next})$ for any $q_{next} \in q_{next}List$
            end
        end
    end
    $Max \leftarrow GetMaxMemoryResourceConstraint(f)$
    return Max
end
```

In parallel composition, given two automata interfaces $\mathcal{I} = \langle X, Y, \xi \rangle$ and $\mathcal{I}' = \langle X', Y', \xi' \rangle$. The automaton interface $\mathcal{I}||\mathcal{I}'$ is a triple $\langle X \cup X', Y \cup Y', \xi'' \rangle$. According to Definition 9, if $\mathcal{I}||\mathcal{I}'$ implements in a system, it consumes a memory resource depending on a pair of μ, μ' for which an assignment over *input/output* of $\mathcal{I}||\mathcal{I}'$ satisfies μ, μ', i.e. $\mathcal{V}(X \cup X', Y \cup Y') \models \xi''$, where $\mu \in \mathcal{M}$ and $\mu' \in \mathcal{M}'$.

Lemma 3. *Given $\mathcal{I}, \mathcal{I}'$. The memory resource consumption for parallel operation has been estimated iff automaton interface $\mathcal{I}$ composes with $\mathcal{I}'$ in parallel.*

Algorithm 3. The computation of memory constraint in Parallel

```
Input: $M = \langle Q, X, Y, q_0, T, l_s, l_t \rangle$ and $M' = \langle Q', X', Y', q'_0, T', l'_s, l'_t \rangle$.
Output: Memory capacity in Max variable.
begin
    Let $f \subseteq Q' \times Q$, $f \leftarrow \{(q'_0, q_0)\}$ and $(q'_0, q_0)$ is unmarked.
    while (true) do
        if GetAllMarked(f)= true then
            return true
        else
            $CurrentElement \leftarrow GetUnmarked((q', q) \in f)$
            let $l_s(q) = p \vdash (R, C)$ and $l'_s(q') = p' \vdash (R', C')$.
            $p'_{next}List \leftarrow GetReachable\mathcal{I}'(Q')$
            $p_{next}List \leftarrow GetReachable\mathcal{I}(Q)$
            $F_{(q_{next}, q'_{next})} : := p \wedge p' \Rightarrow R \wedge R'$
            if $(F_{(q_{next}, q'_{next})} = false)$ then
                return false
            else
                $f \leftarrow (q_{next}, q'_{next})$ for any $q'_{next} \in p'_{next}List$
            end
        end
    end
    $Max \leftarrow GetMaxMemoryResourceConstraint(f)$
    return Max
end
```

Similar to the parallel composition, the following result computes the memory constraint for sequential compositional operation.

Lemma 4. *Given $\mathcal{I}, \mathcal{I}'$. The memory resource consumption for sequential operation has estimated iff automaton interface $\mathcal{I}$ composes with $\mathcal{I}'$ in sequence.*

To bring the result to the practice, the paper introduces Algorithm 4 to calculate the memory resource for the interface sequential compositions.

Algorithm 4. The computation of memory resource constraint in sequential composition

```
Input: $M = \langle Q, X, Y, q_0, T, l_s, l_t \rangle$ and $M' = \langle Q', X', Y', q'_0, T', l'_s, l'_t \rangle$.
Output: Memory capacity in Max variable.
begin
    Let $f \subseteq Q' \times Q$, $f \leftarrow \{(q'_0, q_0)\}$ and $(q'_0, q_0)$ is unmarked.
    while (true) do
        if GetAllMarked(f)= true then
            return true
        else
            $CurrentElement \leftarrow GetUnmarked((q', q) \in f))$
            let $l_s(q) = p \vdash (R, C)$ and $l'_s(q') = p' \vdash (R', C')$.
            $p'_{next}List \leftarrow GetReachable\mathcal{I}'(Q')$
            $p_{next}List \leftarrow GetReachable\mathcal{I}(Q)$
            $F_{(q_{next}, q'_{next})} : := p \wedge \exists Y.(R \wedge p' \wedge l_\theta) \vdash (R \wedge R' \wedge l_\theta \wedge p')$
            if $(F_{(q_{next}, q'_{next})} = false)$ then
                return false
            else
                $f \leftarrow (q_{next}, q'_{next})$ for any $q'_{next} \in p'_{next}List$
            end
        end
    end
    $Max \leftarrow GetMaxMemoryResourceConstraint(f)$
    return Max
end
```

Theorem 1. *The memory resource consumption of a component-based system has estimated iff the systems are constructed by pluggable, parallel and sequential composition of relational interfaces with memory design resources.*

5 Conclusion

The paper carries out a sequential works, from specification and modeling component-based systems by memory resource designs to the construction algorithms in order to estimate memory resources of a component-based system. The paper extends relational interfaces and timed design for estimating memory resource purpose. The memory resource design, which has supreme memory, is an atomic element in an interface, from which we use memory resource design constructing interfaces and environments. The paper also uses labeled automata to model interfaces and environments, and considers some their operations. Using this model, some operations of component-based system are considered such as, refinement, pluggability and compositions. The combination of an interface with its environment and interfaces with the others have made an utilizable method to deal with the specification and modeling problem in estimation of memory resources. Depending on the models, the paper proposes some algorithms to calculate memory resources in the cases of pluggability and composition. According to the approach in this paper, others resources can be specified and be forecast in early stage of the system design.

References

1. Dang Van, H., Truong, H.: Modeling and specification of real-time interfaces with UTP. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 136–150. Springer, Heidelberg (2013)
2. Eskenazi, E., Fioukov, A., Hammer, D., Chaudron, M.: Estimation of static memory consumption for systems built from source code components. In: 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems (2002)
3. Fioukov, A.V., Eskenazi, E.M., Hammer, D.K., Chaudron, M.R.V.: Evaluation of static properties for component-based architectures. In: Proceedings of 28th EUROMICRO Conference, Component-based Software Engineering Track, pp. 33–39. IEEE Computer Society Press (2002)
4. Jonge, M.D., Muskens, J., Chaudron, M.: Scenario-based prediction of run-time resource consumption in component-based software systems. In: Proceedings of the 6th ICSE Workshop on Component-based Software Engineering, CBSE6. IEEE (2003)
5. Muskens, J., Chaudron, M.R.V.: Prediction of run-time resource consumption in multi-task component-based software systems. In: Crnković, I., Stafford, J.A., Schmidt, H.W., Wallnau, K. (eds.) CBSE 2004. LNCS, vol. 3054, pp. 162–177. Springer, Heidelberg (2004)
6. Tripakis, S., Lickly, B., Henzinger, T.A., Lee, E.A.: On relational interfaces. In: Chakraborty, S., Halbwachs, N. (eds.) EMSOFT, pp. 67–76. ACM, New York (2009)