A tool for scalable verification of multi-component distributed systems

Mingyu Park' and Yunja Choi'
'Dept. Of Computer Science and Engineering, Kyungpook National University, South Korea
pqrk8805@gmail.com

Abstract. Composition and visualization are necessary activities for the validation and verification of interaction behavior among distributed system components. This work developed an automated tool support for efficient composition and visualization of component-based systems. The tool automatically constructs an abstract component and its composition behavior from multiple components with published behavior, applies optimization techniques, such as the service-based abstraction method and the conversion technique from non-deterministic finite automata to deterministic finite automata, to reduce the exponentially growing number of states in the successive composition. The tool also helps to reduce potential faults in the composition model through visualizing the constructed composition model. We experimentally show that the optimization method achieves up to 95% of reduction in the number of states in the composition model.

1. Introduction

Distributed embedded systems often consist of multiple components with independent behavior. The overall system behavior is determined by the interaction behavior among those components, which needs to be validated and verified w.r.t the system requirements. For example, each system component may be implemented using virtual prototyping and the overall system behavior is determined by the composition of all the prototype components[1]. In this case, we not only need to validate and verify each virtual prototype, but also do the same for the entire system.

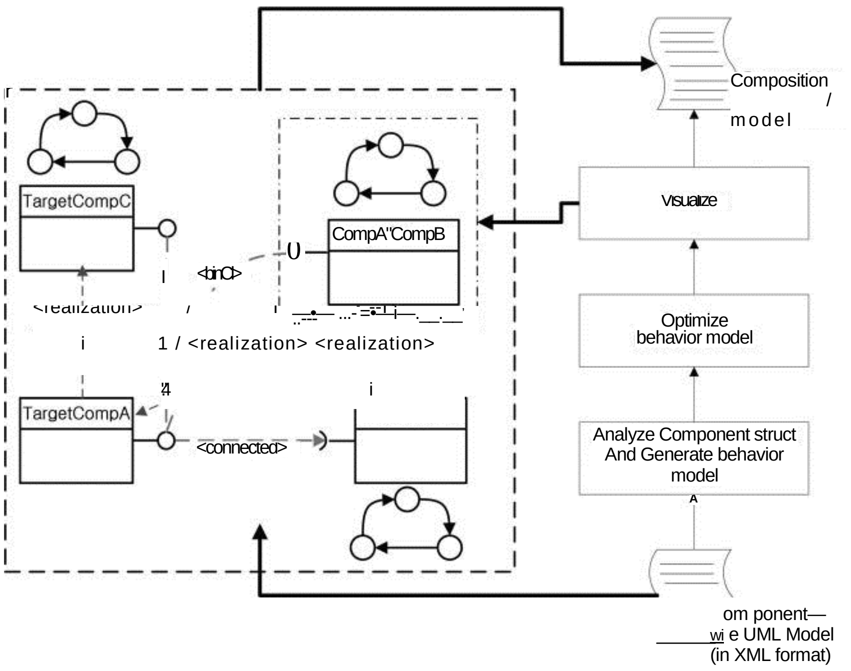
In a bottom-up development, where each component is first developed and composed to implement the system, specifications for the structure and the behavior are available only for each component, but not for compositions. Specifications in this case can be a source code, UML diagrams, or contracts written in formal languages. This makes analysis and comprehension of the overall system behavior quite difficult, as the number of components participating in the composition increases. Simulation is a good way to validate the composition behavior, but its usability is limited by the complexity of the composition; it is difficult to correctly comprehend system behavior when it is a composition of dozens of components.

For efficient and accurate validation and verification of component-based systems, a technique for explicitly representing compositions with respect to their structure and behavior is necessary for developers to intuitively comprehend the system behavior without necessarily looking at the details of each component. Such a method not only helps developers to understand the system better, but also assists in efficient validation and verification as the complexity of the target model is reduced. The necessary technique to achieve this includes automatic construction of the composition model, its visualization, and optimization.

This work provides a tool to construct a composition model; starting from typical parallel composition of unit components, the tool applies optimization methods to reduce the complexity of the composition model. The resulting composition model is visualized as UML class diagrams and state diagrams.

We introduce the composition approach, the optimization methods, and the mapping between UML diagrams and composition model for the visualization. The tool was applied to components of wireless sensor network showing that it achieves maximum 99.5% of reduction of state space, in comparison to the typical parallel composition approach.

2. Approach



Composition model
Visualize
Optimize behavior model
Analyze Component struct And Generate behavior model
Component-wise UML Model (in XML format)
TargetCompC
TargetCompA
CompA"CompB
<bind>
<realization>
<realization> <realization>
<connected>

[Figure 1] Process for automatic construction of composition model

Figure 1 is an overall process of our approach. The input of this tool is a set of unit component models specified in UML and represented in XML format. The tool analyzes the structure of unit components, with respect to their realization relation, extracts target components which participate in a requested composition. Those extracted unit components are composed in parallel and then optimized using abstraction. The final composition model, which is specified in UML and represented in XML format, is used for visualization using UML modeling tool such as Rhapsody and for verification using model checking.

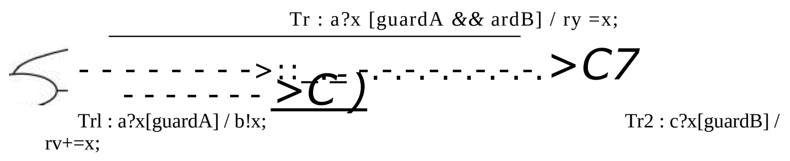
1) Analysis of Component Structure and Generation of behavior model

We initially generate the behavior model of a set of components participating in a composition using typical parallel composition. This requires identifying the set of participating components for the composition from the structural relationship among components.

For example, suppose that we want to compose *TargetCompA* and *TargetCompB* in Figurel. Since *TargetCompA* has no behavior model and all received requests are to be passed through *TargetCompC* due to the bind relation, *TargetCompC* is required to realize *TargetCompA.* Therefore we include *TargetCompC* as a participating component instead of *TargetCompA.* The relationship between *TargetCompA* and *TargetCompC* is a realization relation. We identify participating components by analyzing the realization relation, extract necessary components for a composition with respect to the realization relation, and generate the behavior model using parallel composition of all participating components in this step.

2) Optimization of Behavior Model

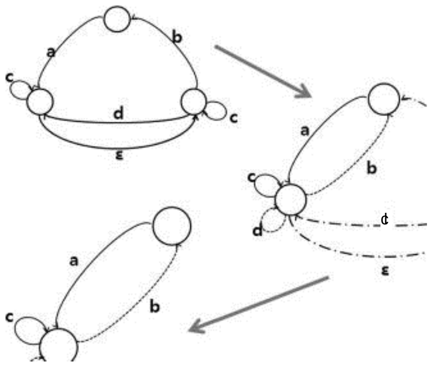
When a component with m states and a component with n states are composed in parallel composition, the behavior model contains m x n states in the worst case. Thus, it can be expected that the number of states would exponentially increase as the number of components to be composed increases in parallel composition. Therefore, the optimization of generated behavior model is essential to reduce the complexity. The second step abstracts the behavior model generated from the first step using the service-based abstraction method[2], and then optimizes the behavioral model by converting non-deterministic finite automata(NFA) - which is generated in the process of the abstraction - into deterministic finite automata (DFA) to reduce the size of behavior model.


Tr : a?x [guardA && ardB] / ry =x;
Trl : a?x[guardA] / b!x; rv+=x;
Tr2 : c?x[guardB] /

[Figure 2] Abstraction of service-based behavior model

A service-based abstraction is conducted by abstracting behavior model based on services to reduce the size of the behavior model. When we verify composite component, it is required to check the correctness of the request/response in composite component. Since interactions among participating components are out of concern in composite component, we can apply service-based abstraction method in figure 2. It means, we can abstract the behavior model of a composition by abstracting and removing service calls among participating components

When a behavior model of composite component is abstracted with service-b ased abstraction, a NFA model including E-transition can be created. Those red undant transitions and states are reduced by transforming NFA to DFA and me rging mergeable states by c-transitions.



[Figure 3] Merging mergeable states by c-transition

3) Visualization of behavior model

When a developer needs to understand the overall behavior of connected components, it is required to observe the whole behavior model of participating components, which is often not a trivial task to do due to its high complexity. Incomprehensible composition behavior may confuse developers and make them contribute to system faults. Intuitive visualization of the behavior model helps to avoid such mistakes.

We transform the generated component model to UML models and then convert them in XML format in the visualization step. In the process of transformation from component model into UML model, component is expressed by Class, Component port by Class port, Component behavior model by state machine and the variables of behavior models by attributes of Class. Finally, updated UML model is converted into XML format. Component structure model is specified in UML Class diagram and the behavior model of the component is specified in state diagram. Developers can easily visualize the generated XML file using UML modeling tools such as IBM Rhapsody.

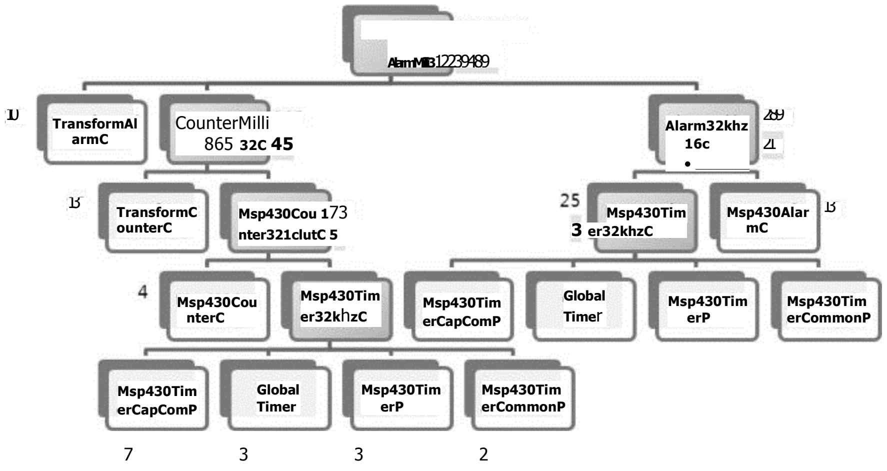
3. Related work

To verify the interaction behavior among distributed system components, Gossler[3] purposed flexible composition method to compose components. But this method did not consider the scalability of the verification. Nejati[4] purposed checking similarity of target statecharts and merging statecharts to generate behavior model. But this method is using heuristic decision, so it is hard to automate. In another approach, Levis[5] developed a tool to simulate TinyOS application, but this approach is difficult to understand interaction behavior among components when it simulates dozens of components.

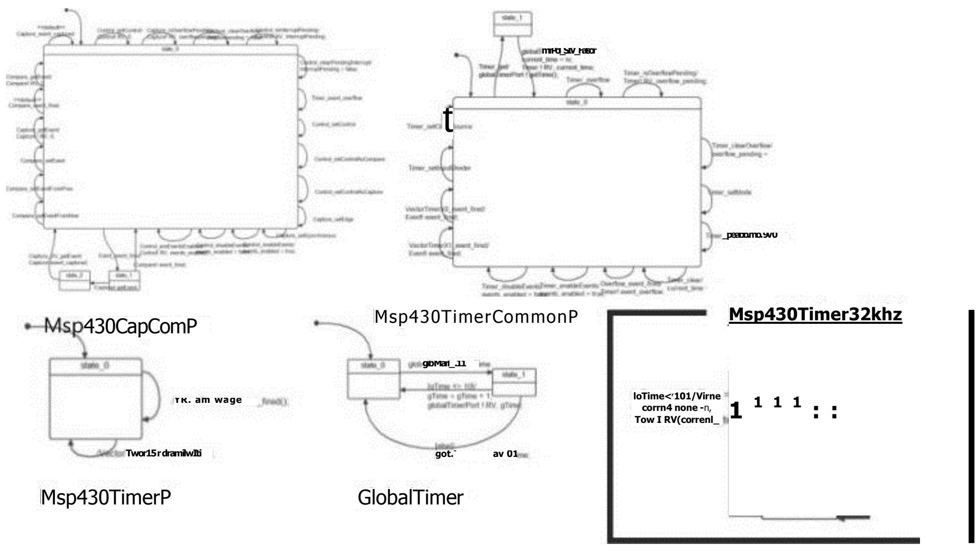
We used component model and service-based abstraction method which is purposed by choi[2] and this tool is the implementation and expansion of it.

4. Experiment

This section describes the experimental result from applying the proposed to of to components of TinyOS[6]. Figure 4 shows the difference in number of states for two cases — one with typical parallel composition and the other after applying abstraction and optimization. The hierarchy in the tree structure repres ents realization relation.



[Figure 4] Number of states generated for composition models



[Figure 5] Visualization after composition

We note that the numbers of states are dramatically decreases as the composition continues towards the root of composition tree, compared to those from typical parallel composition. No exponential increase was found after optimizing behavior models, and in some cases, we found it was smaller number of states than before composition — such as Msp430CounterC and Msp430Timer32khzC. In case of a topmost component AlarmMilli32C, the behavior model only has 1,202 states. This number of states is merely 0.05% of parallel composition method.

Figure 5 is an example visualization of the behavior model before and after composition. We need to follow all 4 state diagrams before the composition and visualization to understand their interaction behavior. In contrast, the behavior model that presents interaction among the four components can be seen clearly after visualization. Also, the size of system that needs to be identified became much smaller, and consequently shows that this visualization helps comprehending the behavior among components by optimization.

5. Conclusion

This work developed a tool for automated composition and visualization of components in order to verify interaction behavior of distributed, multi-component systems. The tool generates behavior model for an arbitrary composition, performs abstraction and optimization to reduce the complexity of the composition model. We showed that the optimization can result in 95% reduction of the number of states required to specify the same behavior model. The composition model is visualized by converting it into XML format which can be imported by UML modeling tools.

Acknowledgement
This work was supported by the IT **R&D** program of MKE/KEIT. [10041145, Self-Organized Software-platform (SOS) for welfare devices].

Reference

[1] James C. Schaaf, Jr., Faye Lynn Thompson. "Systems Concept Development with Virtual Prototyping" Proceedings of the 29th conference on Winter simula tion, pp. 941 - 947. 1997.
[2] Yunja Choi, Moonzoo Kim. "Controlled composition and abstraction for bottom-up integration and verification of abstract components". Information & Software Technology 54(1), pp. 119-136. 2012
[3] G. Gossler, J. Sifakis. "Composition for Component-Based Modeling," Science of Computer Programming, vol. 55, pp. 161-183.2005.
[4] Shiva Nejati et al. "Matching and Merging of Statecharts Specifications" ICSE '07 Proceedings of the 29th international conference on Software Engineering, pp. 54-64. 2007
[5] Philip Levis, Nelson Lee, Matt Welsh, David Culler. "TOSSIM: accurate and scalable simulation of entire TinyOS applications". SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems, pp. 126-137. 2007
[6] TinyOS. <http://www.tinyos.net/>