Towards a Formal Model for Reconfigurable Software Architectures by Bigraphs

Zhiming Chang, Xinjun Mao, Zhichang Qi
Dept. of Computer Science, National University of Defense Technology, China, 410073
zmchang@nudt.edu.cn

Abstract

With the spread of the Internet and software evolution in complex intensive systems, software architecture often need be reconfigured during runtime to adapt variable environments and design objectives. To deal with reconfigurable software architectures, the formal method should be presented to describe software architectures and express their changes so that these changes on the evolutions of software architectures could be reasoned about. However, current formal methods for reconfigurable software architectures are difficult to represent hierarchy and model context-aware systems. In this paper, we use and extend Bigraph as a formal method to describe reconfigurable software architecture. By providing graphic elements and term languages, extended bigraphs can survey static and dynamic architectures easily. Then we represent basic architectural operations on extended bigraphs, through a case describe reconfigurations with constraints and context-aware information by reaction rules, and illustrate how to check the properties to satisfy design requirements by BiLog.

1. Introduction

Nowadays, increasingly systems are required to run continuously. Moreover they must often do this in the context of environments whose resources are constantly changing [1]. These software systems are required to be able to adapt themselves at runtime to handle changes of variable environments. As the blueprints of software systems, software architectures can provide a global perspective of the systems and expose important system-level properties and integrity constraints. Obviously, software architectures should eventually reflect the changes through architectural operations such as adding/removing/replacing components/Connectors or connecting/disconnecting. Study on adaptations to dynamic software architecture at runtime will benefit greatly on the improvement of the capability of self-adaptive software systems and has been paid more and more attention in the software engineering community.

To deal with reconfigurable software architectures, the formal method should be presented to describe software architectures and express their changes so that these changes on the evolutions of software architectures could be reasoned about. After that, the new architectures can be produced by combining changes with the original architectures together, and checked whether they can preserve the constraints of the desired software architectures by analysis and verifications.

Although some formal methods for reconfigurable architectures have been done to address these issues, they lack of mechanisms to describe contexts, constraints, and hierarchies. Most of them can’t be understood easily, and difficult to validate the design principles. In this paper, we use extended bigraph as a formal method to describe reconfigurable software architectures, where bigraphs, operations on bigraphs, and reaction rules represent architectures, architectural operations, and architectural reconfigurations respectively. By appending context-aware information to operations or reaction rules, we can model context-aware evolving architectures. And some design principles can be validated by using BiLog. Therefore, bigraph can provide a high-level view of dynamic software systems intuitively and formally.

2. Overviews of BRS

The theory of bigraphical reactive systems (BRS), due to Milner and co-workers, is based on a graphical model of mobile computation that emphasizes both locality and connectivity [2]. Informally, a BRS consists of bigraphs and a set of reaction rules, where bigraphs can be allowed to reconfigure themselves by reaction rules.

![Figure 1 A bigraph involves two parts: place graph and link graph](image)

In Figure 1, the bigraph $G$ has nodes, indicated by solid
shapes, here is \( \{ v_0, \ldots, v_i \} \). Each node has a control, written in capital letters; here is \( PC, \) \( ROOM, \) and \( PDA. \) Each control has a number of ports; ports can be linked by edges, here is \( \{ e_0, \ldots, e_j \}. \) For example, the control \( PC \) has 2 ports. Nodes can be nested, indicated by containment. The two outermost dashed boxes indicate regions. The two dark solid boxes indicate sites. There is an outer name \( y \) and inner name \( x. \) The bigraph \( G \) involves two parts: place graph and link graph.

Bigraph can be also be expressed by term language, the primary operations and elements used by the paper are summarized in Table 1.

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U</td>
<td>V )</td>
</tr>
<tr>
<td>( U \setminus V )</td>
<td>Composition</td>
</tr>
<tr>
<td>( U(V) )</td>
<td>Nesting. ( U ) contains ( V )</td>
</tr>
<tr>
<td>( K_x )</td>
<td>Node with control ( K ) of face ( x )</td>
</tr>
<tr>
<td>1</td>
<td>The barren region</td>
</tr>
<tr>
<td>( \mathcal{E} )</td>
<td>The empty wiring</td>
</tr>
<tr>
<td>( _ )</td>
<td>Site numbered ( i )</td>
</tr>
<tr>
<td>( U, x )</td>
<td>Closure of inner name ( x ) in ( U )</td>
</tr>
<tr>
<td>( \langle x, U \rangle )</td>
<td>( U ) with outer name ( x ) replaced by an edge</td>
</tr>
<tr>
<td>( x/Y, U )</td>
<td>Connection inner names ( Y ) to outer name ( x ) in ( U )</td>
</tr>
</tbody>
</table>

Each reaction rule consists of a redex, which may be transformed into a reactivity wherever it occurs; both of these are themselves bigraphs. Figure 2 shows that if a \( PC \) in a room is linked to remote \( PC, \) a \( PDA \) in the same room may use the \( PC \) as a gateway to copy data from remote \( PC. \)

![Figure 2 A reaction rule](image)

3. Extended \( \Sigma \)-sorted BRS and Architecture

Due to the space of lines, the extended \( \Sigma \)-sorted BRS can be seen in [3], where we modify control, define face, and extend bigraph and \( \Sigma \)-sorted BRS. An extended \( \Sigma \)-sorted BRS means the bigraphs and reactions rules hold some conditions roughly.

The main elements of software architecture are as follows: components, connectors, configurations, ports, roles, representations, and rep-maps. Table 2 shows the relationship of reconfigurable software architectures and extended \( \Sigma \)-sorted BRS. In [3], extended bigraphs and conditions can be used to describe architectures and constraints respectively. In this paper, due to reconfigurable architecture, we focus on the architectural operations and architectural reconfigurations.

![Table 2 Mapping reconfigurable architecture to \( \Sigma \)-sorted BRS](image)

4. Architectural Operations

Since most of the operations would change the inner and outer interfaces of the bigraph, they can’t be transformed by reaction rules. Hence, we directly modify bigraph to express their operational semantics. The main elements of software architecture are as follows: components, connectors, configurations, ports, roles, representations, and rep-maps. Table 2 shows the relationship of reconfigurable software architectures and extended \( \Sigma \)-sorted BRS. In [3], extended bigraphs and conditions can be used to describe architectures and constraints respectively. In this paper, due to reconfigurable architecture, we focus on the architectural operations and architectural reconfigurations.
also can provide internal and external interfaces that can be analogy with providing an interface and requiring an inter-implementation in software architectures respectively. Hence, we extend four more operations for interfaces.

- Adding an external interface
  \[ \text{AddEface}(P_y, x, SA) = \text{sub}(P_{x,y(z)}, x, SA) \]
- Adding an internal interface
  \[ \text{AddIface}(P_y, x, SA) = /x.\text{sub}(P_{x,y(z)}, P_x, SA) \]
- Removing an external interface
  \[ \text{RemoveEface}(x, SA) = /x.SA \]
- Removing an internal interface
  \[ \text{RemoveIface}(x, SA) = SA.x \]

### 5. Architectural Reconfigurations

While changes occurring in software architectures, they rarely happen independently. That is, changes often result in cascade reactions in order to ensure the integrity of software architectures. For example, when a new component is added in, new connectors often need to be defined to provide connection services for the component. Moreover, these changes are often influenced by variable environments. We call these changes of architecture as architectural reconfigurations.

We use reconfigurable approach to describe architectural reconfigurations; the main steps are illustrated as follows:

- Step 1: Describe the initial architecture by bigraph, and its reaction rules.

  Note that all of the reaction rules must preserve the constraints of the software architecture. It has been proven that the changing bigraphs always preserve the constraints defined by \(\Sigma\)-sorted BRS if the initial bigraph and reaction rules do [3].

- Step 2: Append context-awareness to reaction rules.

  By using information about context, each reaction rule is provided with fire conditions.

- Step 3: Check whether the evolving architecture satisfies design principles by BiLog [4].

Now, we consider a web-based Management Information System (MIS) as a case, in which web users access management services by making requests to geographically distributed servers via the CORE, as seen in Figure 3.

![Figure 3 MIS and an instance described by bigraph](image)

Obviously, in web environment, web clients and servers are not always online and connected with the CORE. Thus, the software architecture of the MIS may dynamically change due to some reasons. For example, web users dynamically log on, web servers need not always be connected with the CORE, these servers are required to connect with the CORE and asked to provide services only when web users have requests.

Since the MIS belongs to client-server style, its constraints are: (1) all the connectors are RPC; (2) any connector RPC has only two roles: caller and callee; (3) the port of the component client should be attached to caller while the port of the component server should be attached to callee. These constraints can be mapped to the conditions of extended \(\Sigma\)-sorted BRS.

![Figure 4 Four context-free rules for MIS](image)

In the first step, we define the initial architecture is just one CORE node, four reaction rules for the dynamic system: “Add a Client”, “Remove a Client”, “Add a Server”, “Remove a Server” in Figure 4, where outer interfaces represent potential linkages. All the four rules preserve the constraints of client-server style. For example, in “Add a Client” reaction rule the adding connector is RPC, the role caller is attached to CLIENT (not shown in the Figure 4), and another role callee is attached to CORE.

Of course, we can also use composition of basic operations to represent the reaction rules. For example, the first rule can be rewritten as follows:

\[
\begin{align*}
&\text{Add a Client} \{ \text{Add(CLIENT request, SA)} \} \\
&\text{Add(RPC callee caller, SA)} \\
&\text{Connect(request, callee, SA)} \\
&\text{Connect(reply, caller, SA)}
\end{align*}
\]

Since the MIS dynamically increase or decrease the number of servers to adapt the requests by web users, we assume that client can join or quit the system at random; every server can provide services for \(N\) clients concurrently at most.

![Figure 5 Four context-aware rules for MIS](image)

In the second step, we use a CONTAINER as context to append every reaction rule in Figure 5. The CONTAINER means what’s the number of clients that the current servers
6. Related Work

Existing work falls into two main categories about architectural reconstructions roughly: architectural description languages (ADLs) and general formal methods. Some ADLs support the description of dynamic aspects of architectures. They are including: Dynamic-ACME [5], C2/AML [6], Darwin, Rapide, Dynamic-Wright [7], π-ADL [8]. Dynamic-ACME and C2/AML have no formalism. But others consider less about changes of environments, less focus on architectural operations and reconstructions explicitly, and their formal methods are not intuitional.

The second area of related work is general formal methods. In [9], it looks at ways to use π calculus to specify dynamic software architectures. However, this approach is unable to describe structural aspect of software architectures. For example, the new adding component can not be nested in another one. In Addition, it doesn’t make use of constraints of these changes to guide how to reconfigure software architectures. In [10], it uses CHAM to model architectures. As a general term rewriting system, CHAM can describe arbitrary architectural reconstructions. Another similar formalism is the use of graph grammars to describe the allowable topologies of architectures [11]. Unfortunately, these two methods have little ability to represent parameters and context-aware systems, for example, passing parameters and global conditions (e.g., “if every client is not connected to the printing systems, then …”) may be impossible or very difficult to specify [10]. Moreover, those two approaches don’t describe some of important architectural elements, such as ports and roles.

7. Conclusions and Future Work

In this paper, we extend bigraph as a formal method to represent basic architectural operations by bigraph operations expressed by term languages; and describe reconstructions with constraints and context-aware information by reaction rules.

However, there are still a number of open problems that should be dealt with and will guide our future work.

− Lack of principles to guide architects to append context-aware information to reaction rules.
− Need to represent behavioral aspect of software architecture.
− Lack of tools to support for validation.

Acknowledgements. The authors acknowledge financial support from Natural Science Fund (Contract No. 60773018), the State 863 High-Tech Program (Contract No. 2007AA01Z135) of China, and Postgraduate Innovation Fund of National University of Defense Technology 2007.

References