International Journal of Automation and Computing

13(6), December 2016, 552-564
DOI: 10.1007/s11633-016-1028-9

xBreeze/ADL: A Language for Software Architecture Specification and Analysis

Chen Li[1] Hong-Ji Yang[1] Mei-Yu Shi[2] Wei Zhu[2]

[1]School of Humanities and Cultural Industries, Bath Spa University, UK

[2]Tourism Institute, Beijing Union University, Beijing 100101, China

Abstract: Architecture description languages play an important role in modelling software architectures. However, many architecture description languages (ADLs) are either unable to deal with the verification and dynamic changes directly or too formal to be understood and manipulated. This paper presents xBreeze/ADL, a novel extensible markup language (XML)-based verification and evolution supported architecture description language, which is specifically designed for modelling the software architecture of large, complex systems. Five principle design goals are 1) to separate template from instance to define a loose coupling structure, 2) to present virtual and concrete link to identify service execution flow, 3) to clearly represent component behaviour to specify architecture semantics, 4) to introduce multi-dimension restrictions to define the architecture constraints, and 5) to use the graph transformation theory to implement the architecture configuration management (i.e., reconfiguration and verification). Various advanced features of xBreeze/ADL are illustrated by using an example on online movie ticket booking system.

Keywords: Software architecture, architecture description language (ADL), xBreeze/ADL, breeze graph grammar, Breeze/ADL.

1 Introduction

With the development of the Internet technology, software development and operation are facing a changing, complex and uncontrollable environment. In order to control the complexity of software systems, people start to transfer their focus to three aspects: software structure, evolvability and quality. Thus, in order to support the software development and quality assurance in a software life cycle, it is an urgent need to study technologies of software specification and analysis.

Software architecture, as the abstraction of the software system, plays an important role in manipulating and analysing a system at a high level, and it represents the complexity in the global view of software systems. Generally, there are three basic elements of the software architectures, i.e., architecture structure (e.g., architecture instance), style (e.g., templates and constraints), and configuration management (e.g., architecture reconfiguration).

Various techniques and tools have been proposed to specify and model the software architecture. One of the most popular ways to describe the software architecture is architecture description languages (ADLs). By using ADLs, the system enjoys a high efficiency of development and its complexity can be controlled. However, most ADLs are either only considering static aspects of the architecture or unable to deal with the change directly. Besides, the service execution flows within a component is not clearly discussed. Moreover, few of them have strong flexibility and extensibility, and can provide multi-dimension constraints to restrict the interaction behaviour, or support architecture verification and transformation explicitly.

To remedy this, we have developed xBreeze/ADL, an eXtensible markup language (XML)-based, verification and evolution supported architecture description language specifically designed for modelling software architecture of large, complex systems. These systems are composted of various components, and require high evolvability to adapt to the change of system or environment. xBreeze/ADL is first developed for specifying and analysing the software architecture of those systems. Based on our former work[1, 2], this new ADL-xBreeze/ADL is 1) to map a software architecture into a graph and the architecture structure is represented in terms of the nodes and edges, 2) to use production, leveraging graph transformation idea, which defines the pre-condition and post-condition of an operation to support the architecture evolution and constraint verification, 3) to specify the component semantic by defining the action of the port for capturing the component′s behaviour, 4) to provide restrictions for constructing behaviour constraints from four aspects (i.e., action dependence, time dependence, state dependence and execution dependence), 5) to define three general reconfiguration operations (i.e., addition, removal and replacement) for augmenting evolution support at the level of components/connectors, and 6) to assign the running pattern (i.e., sequence, parallel and iteration) to the above reconfiguration operations for describing the composite operations.

In this paper, we first list related work (Section 2). We

Research Article
Manuscript received December 4, 2015; accepted April 21, 2016
This work was supported by National Natural Science Foundation of China (No. 71473018).
Recommended by Associate Editor Matjaz Gams

	Wright	Rapide	xADL2.0	ABC/ADL	AGG	Breeze/ADL	xBreeze/ADL
Template	–	–	+	+	–	+	+
Instance	+	+	+	+	+	+	+
Style	+	–	+	+	++	++	++
Behaviour	+	++	+	+	–	+	++
Connection	–	+	+	+	+	+	++
Constraint	+	+	–	–	++	++	++
Reconfiguration	–	–	–	–	+	+	+
Running pattern	–	+	–	–	–	–	+

Fig. 1 Feature comparison of related architecture description techniques

survey several popular ADLs which are related to the xBreeze/ADL and summarize the common features of them. The overview of xBreeze/ADL is described in Section 3. The key features of xBreeze/ADL include the Architecture Template (Section 4.1), Semantic (Section 4.2), Link (Section 4.3), Style (Section 4.4), Constraints (Section 4.5), Instance (Section 4.6), Rule (Section 5.1) and Configuration (Section 5.2). An online movie ticket booking system has been chosen in order to show flexibility and extensibility of xBreeze/ADL to practical problems in Section 6. Section 7 concludes the paper.

2 Related work

As a blueprint of the software systems, software architecture plays an important role in depicting the skeleton of the whole system, especially in large and complex systems, in a global view. A series of design decisions can be completely illustrated through the software architecture. Comparing with lines-of-code, coarser-grained architecture elements and their interactions illustrate a high level abstraction of the system. It provides a better manner of understanding and manipulating a system. To support the software architecture development, many ADLs have been proposed.

One of the most popular techniques for describing a software architecture is the ADLs, like Wright[3], Dynamic Wright[4], Aesop[5], ArTek[6], C2[7], MetaH[8], LILEANNA[9], SADL[10], Weaves[11], Darwin[12], Rapide[13], Unicon[14], ACME[15], xADL 2.0[16], ABC/ADL[17], etc. Those ADLs depict the abstract information of the software systems at a high level, e.g., Style, behaviour and Configuration.

Wright is proposed to support a static software architecture, and its formal syntax basis is communicating sequential processes (CSP)[18]. It is often used in correctness verification such as deadlock analysis and consistency verification, but it cannot support a dynamic software architecture. Darwin is an ADL which focuses on three atomic services-provide service, request service and binding service, but Darwin cannot specify connectors explicitly and independently. Rapide is developed to describe the computation behaviours and interaction behaviours, using the partially ordered event sets, but it cannot fully support the expression of the interaction models of components. Unicon is proposed by Shaw et al., which uses Role to stand for participants of an interaction and player to stand for interfaces of components, but it′s insufficient in the specification of the software architecture style. ACME is proposed by Garlan et al., whose purpose is to provide a simple and general ADL. Technically, ACME is preferred to be an architecture transformation language instead of ADL. It provides a mechanism for the transformation between different ADLs, and also supports an open semantic framework for labeling the attributes of the architecture structure. xADL 2.0, like our xBreeze/ADL, is an XML-based ADL. Based on xArch, which provides a core specification for basic architecture elements, xADL 2.0 supports run-time and design time modelling and uses the XML schema to extend the core. Unfortunately, xADL 2.0 cannot fully implement the architecture behaviour and constraints definition, and the style verification only can be done on C2 style architecture. ABC/ADL is also an XML-based ADL. ABC/ADL combines the top-down and down-top views together to model the software architecture. It uses two-layer mapping mechanisms, i.e., 1) architecture modelling results to UML and 2) UML to code framework, to deliver the basic code framework of an architecture to the developer. However, it also fails to perform the architecture style verification, reconfiguration and constraints definition. To remedy this, people try to use some graph grammars, like attributed graph grammar (AGG)[19], for software design and development. These formal graphical notations not only help modelling and representing the software system but also provide a vivid way to achieve the system analysis and evolution. However, they do not support architecture design directly, e.g., the interface or connector cannot be found in the definition of graph which is one of the most important characteristic of a software architecture. Breeze/ADL[1], one of our former work, is an architecture language which adopts XML as the meta-language and has the ability of describing the software architecture. Breeze/ADL specifies the software architecture in an XML format and captures the change during both initial development and subsequent evolution by performing production according to the Breeze graph grammar[2] productions. However, Breeze/ADL still needs to be improved regarding the behaviour definition, multi-dimension restrictions specification, hierarchical composition, reconfiguration refinement and running pattern definition aspects, and that is the motivation behind developing xBreeze/ADL. The aims of xBreeze/ADL are to capture the component/connector′s semantic, verify the style/constraints and achieve evolution by using architecture as a framework for integrating series of components, connectors and connections into software systems.

Most of the above proposed ADLs only focus on the static aspect of architecture, e.g., they are unable to deal with the change directly, especially for dynamic evolution of software systems. Also, these ADLs cannot fully support the architecture style verification, behaviour specification and constraints definition. Our paper builds upon the results of the above efforts, e.g., borrowing some ideas of Rapide to enhance the semantic and constraint expressing, leveraging graph transformation to implement the reconfiguration. Fig.1 shows a feature checklist for above the architecture description techniques and our xBreeze/ADL. The "–", "+" and "++" represents the feature is not supported, supported and well supported, respectively.

3 Overview of xBreeze/ADL

An xBreeze/ADL document is an XML document that follows a set of xBreeze/ADL schemas. These schemas are classified into two groups according to their purposes. One set of schemas establishes the architecture modelling and the other supports for the architecture configuration management. Modelling features provided by schemas are described in Fig. 2.

3.1 Architecture modelling

xBreeze/ADL specifies the software architecture in terms of architecture templates, styles, constraints and instances. Seven schemas are involved into architecture modelling: 1) the Template schema for describing the primary elements, i.e., components, connectors and ports, 2) the Link schema for specifying the connection among the components/connectors, 3) the Semantic schema for modelling the behaviour of components, 4) the Style schema for providing the architecture style with defined templates and style constraints, 5) the Constraint, 6) Rule schemas for writing the abstract restriction (e.g., behaviours constraints, style constraints), and 7) the Instance schema for describing the architecture instance according to the style and constraint.

xBreeze/ADL maintains a separation between the template and instance of an architecture model. In most cases, a software system may contain different subsystems which might follow various styles and constraints. Therefore, supporting independent template definition will contribute to implement different sub-architectures with different styles. In general, most ADLs precisely define the external interaction connections since the component is usually considered as a black-box. The benefit of doing this is to simplify the architecture design process by hiding the internal communication of the components. Though this helps to identify the service execution flows among the components, it is unable to trace the invocation paths from the provider interfaces to the requestor interfaces within the component. xBreeze/ADL not only concerns external interactions among components but also considers the internal communications within a component, with the help of Link schema. xBreeze/ADL supports the architecture semantic specification by modelling the component′s behaviour which is captured through the interactions among the interfaces.

xBreeze/ADL, using predefined templates, maintains architecture style information. It also uses the multi-dimension constraint and Production which is defined in rule schema (see Section 5.1) to specify the behaviour constraint and style constraints respectively. Generally, an architecture instance can be generated if all of its instances (including component instances and connector instances) subject to the corresponding templates and constraints.

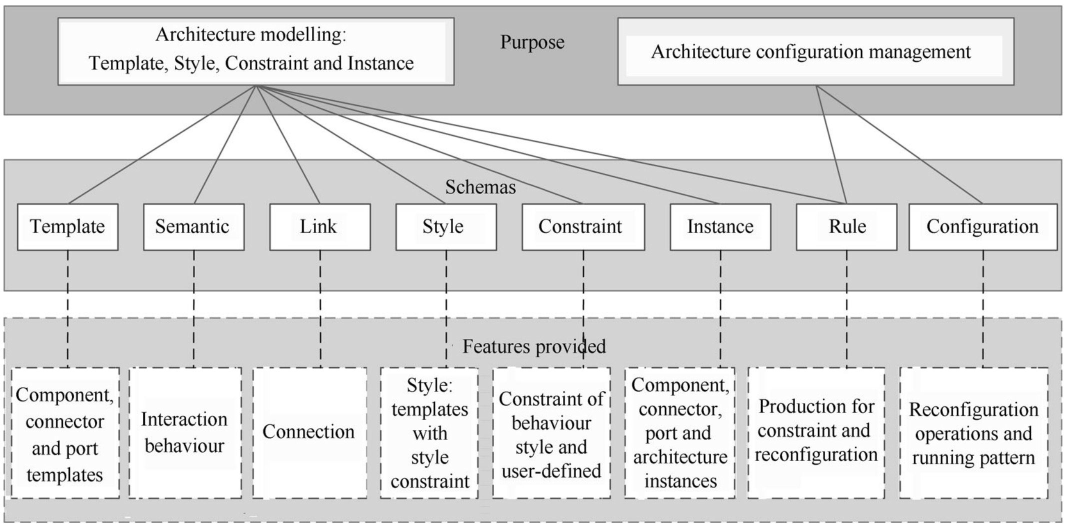

Architecture modelling: Template, Style, Constraint and Instance
Purpose
Architecture configuration management
Schemas
Template
Semantic
Link
Style
Constraint
Instance
Rule
Configuration
Features provided
Component, connector and port templates
Interaction behaviour
Connection
Style: templates with style constraint
Constraint of behaviour style and user-defined
Component, connector, port and architecture instances
Production for constraint and reconfiguration
Reconfiguration operations and running pattern

Fig. 2 xBreeze/ADL schemas and features

3.2 Architecture configuration

One of the essential modelling features of software architecture is dynamic evolution. In order to support the architecture evolution, xBreeze/ADL uses two schemas, Rule and configuration schemas, to implement the architecture reconfiguration and define their running patterns. xBreeze/ADL borrows the idea of graph rewriting rules of Breeze graph grammar to define the architecture transformation with Rule schema. Recall that the style constraints verification is also specified with the help of Rule schema. Unlike other ADLs, xBreeze/ADL explicitly defines three reconfiguration operations, i.e., Addition, Removal and Replacement, to support the architecture evolution with configuration schema. Moreover, xBreeze/ADL also allows architects to arrange the running pattern of the reconfiguration operations. To be specific, those operations can be executed in a pattern (Sequence, Parallel or Iteration).

4 Architecture modelling schemas

In this section, we give a brief overview of each of the schemas for modelling the software architecture.

4.1 Template schema

xBreeze/ADL makes explicit distinctions between the architecture template and instance. Like the Class in object-oriented design (OOD), a template provides a framework or pattern for the architecture elements. The template schema specifies the node template and super node (i.e., composite nodes or nested nodes) template for a software architecture. Using nodes and edges to build an architecture is based on the idea of Breeze graph grammar and it provides a vivid way to illustrate a software architecture.

xBreeze/ADL uses the node template to support the definitions of both component templates and connector templates. Each node template provides the basic information for its instance node(s). An instance node satisfies a node template if it defines all items declared in the template. In general, each instance node is generated from a node template, and instances may be derived from the same template. Some key features need to be specified in a component or connector template, i.e., Type, Class and Port template. Fig. 3 shows the essential meta elements which the template schema need to specify.

The Type attribute helps define the type of the node template which could be a component or a connector. Component, as a functional part of a software system, plays a key role in providing services for users. The functions of a component are usually defined in Class(es) which help(s) people to identify and understand the components.

Thus, xBreeze/ADL requires the node template to declare its Class attribute explicitly when it is generated. Given the Class for the template, all instances of this template will have the same Class which means they all share common functions and follow the same patterns. Moreover, separating the template from the instance makes the instance not only inherit the functions of its template but also be capable of being extended to support new functions. This OOD feature makes the xBreeze/ADL enjoy a high extensibility and flexibility.

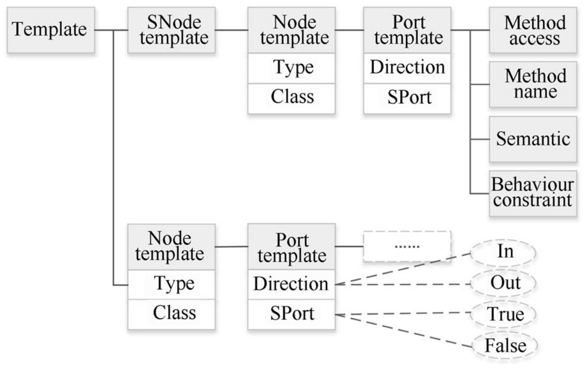


Fig. 3 Template schema: defining template for super node and node.

Port, as one of key elements of a component or a connector, is responsible for interactions among components. The main functions of a component or a connector are defined in its ports. A port template contains three elements: method access type, method name, semantic (Section 4.2) and constraint (Section 4.5). The SPort attribute indicates that the port is a regular port (if its value is set to false) or a super port (if its value is set to true). xBreeze/ADL uses Direction attribute to identify a requestor port (i.e., direction is “In”) or a provider port (i.e., direction is “Out”). A component uses its requestor ports to ask other components for their services. Thus, the requestor ports are usually declared as private or protected, i.e., these ports are only visible within the component or the subcomponent. On the contrary, the provider ports are those defined in public, which means other components may invoke them.

One of essential features of xBreeze/ADL is supporting the hierarchical composition by capitalizing the super node. Each super node, representing a composite node or a subsystem, contains nodes and edges, and might have its own style. The SNodeTemplate encapsulates the node templates which will be used to construct a composite node template to meet the system requirements. The super node uses some ports from its sub nodes to interact with other node, and the SPort attribute helps to declare such super port templates for the super node template.

The meta elements of the template schema presented here only provide essential features for building the particular type of instances.

4.2 Semantic schema

xBreeze/ADL explicitly specifies the component′s semantics in terms of behaviours of the port. As mentioned above, the port template specifies the semantics of the component by semantic schema which defines essential meta elements of the behaviour.

To model the dynamic component behaviour, xBreeze/ADL abstracts the interaction information among the components, and separates service requestor behaviours and service provider behaviours according to the method which is defined in the port template. The semantic schema uses a requestor element to encapsulate two actions: Caller and Return for a requestor port template. The Caller action needs to specify the method parameters (MethodParameter) which will be sent to the port template of the service provider. The requestor port template also uses its Return action to describe the type of result (MethodReturn) which is expected by the service provider. It is similar to the provider port template which capitalizes the Provider element to define two actions: Callee and Result. The Callee action depicts the requirements of the type of the method parameters which are expected by the service requestor. The Result action tells the requestor port what type of the result the provider will send. Fig. 4 shows the essential meta elements which the semantic schema specifies.

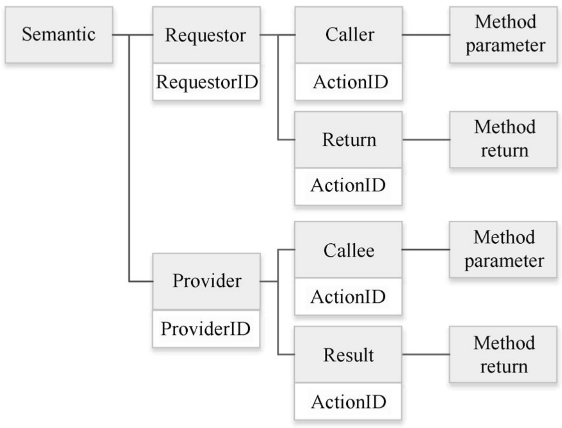


Fig. 4 Semantic schema: defining semantics in terms of behaviours

4.3 Link schema

Unlike other ADLs, xBreeze/ADL explicitly defines the link (i.e., edge element) for two purposes.

The first is refining the connections. Edge defines two elements, i.e., Source and Target, for a connection (see Fig. 5). The Source indicates the component which starts an invocation, and the invoked component is the Target. The edge not only helps to bind ports (e.g., connecting the service requestor port to the service provider port) between the components but also supports the behaviour binding (e.g., connecting the action of service requestor port to the action of a service provider port). xBreeze/ADL brings more clear communication channels to the software architecture, and the behaviours among the components are more easy to be understood.

The other is identifying the implicit service execution flows within a component. Most ADLs do not support the implicit invocation path within the component. If a component has several requestor and provider ports, those ADLs might help to understand what functions are provided or requested by those ports. That means it only shows external interaction behaviours of a component, but internal communications among these ports within the component cannot be learned even at an abstract level. Thus, xBreeze/ADL uses the Virtuality attribute of the edge to identify the links. To be specific, xBreeze/ADL provides two types of edge, virtual edge and concrete edge. In general, the edge among the components is called a concrete edge, i.e., the value of its Virtuality is set to false, and the interaction among ports within a component refers to virtual edge in which Virtuality is true. Unlike edges among components, the virtual edges only provide an abstract information among ports. That means they only need to bind the ports together to help developers to understand the service execution flows within a component.

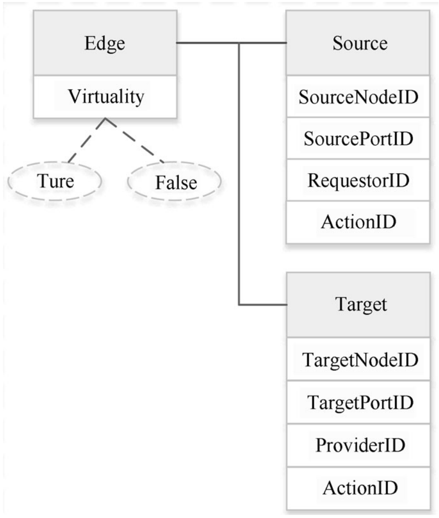


Fig. 5 Link schema: defining a connection between service provider and requestor

4.4 Style

Style is a skeleton of the software architecture and it guides the developer how to build an architecture instance. As we mentioned before, to describe an architecture style, we need to introduce the related component and connector templates, semantics and links, and bind them to certain style constraints (Section 4.5).

Since xBreeze/ADL supports the hierarchical composition, a composition node (i.e., super node) might have its own style. Thus, developers may define different styles for the software architecture. Each style only contains necessary component and connector templates and uses edges to bind them together to form an architecture reference model[20]. Fig. 6 shows the meta elements of the style schema.

4.5 Constraints

xBreeze/ADL supports specifications of architecture constraints in three aspects: behaviour constraints, style constraints and user-defined constraints.

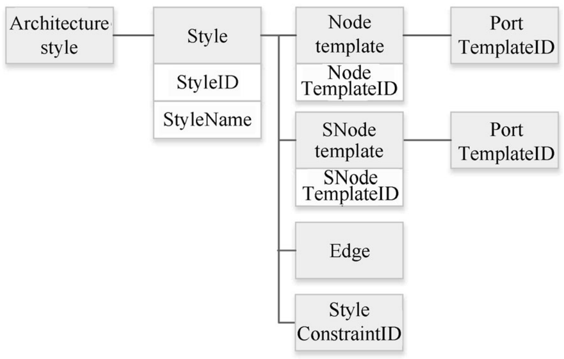


Fig. 6 Style schema: defining an architecture skeleton and constraint

To precisely define the behaviour constraints, xBreeze/ADL concerns four dimension restrictions of a behaviour, i.e., action dependence, time dependence, state dependence and execution dependence. Action dependence helps to identify which behaviour should happen before others. Given the action dependence, the interaction restrictions are easy to be understood, and this also helps to identify the service execution flows among components. For example, a response action always happens after a request action. Action dependence defines Precondition element and Postcondition element for Pre-action and Post-action respectively. To support the dynamic change of the software architecture, xBreeze/ADL introduces the time factor into component behaviours and binds the actions with time restrictions. A component may show different behaviours time to time according to the time dependence. There are three time periods that we concern in time dependence, i.e., Between, Before and After. The Between element capitalizes the Start and End attributes to specify a time interval for an action, which means the action is only valid during this time period. The Before and After elements depict an action that should start before and after a specific time respectively. During the system running time, known or unknown factors (e.g., internal bug, unstable environment) might lead components to a failure state. Such state should be reflected to the architecture level for further analysis and design. Therefore, xBreeze/ADL focuses on four types of states: Active (means working), Idle (waiting for invoking), Suspend (cannot be visited temporarily) and Failure (stop working due to some errors). Binding state with behaviour helps to obtain more running details of the component and also to restrict the component′s behaviour. Actions within (among) the component(s) might execute concurrently. xBreeze/ADL borrows some ideas of current systems to support a Parallel and Sequence execution pattern for component behaviours. Fig. 7 shows the meta elements of the Constraints schema.

Style constraints define basic restrictions among the templates. Unlike behaviour constraints, it focuses on the way to connect templates instead of the specific behaviour of the component. In general, each style has its own constraints. For example, Client/Server style requires the communications that only happen between the client component and server component. xBreeze/ADL also supports user-defined constraints at the architecture level. Designers may set the restrictions for the architecture structure (without violating the style constraints), behaviour or attribute. Benefiting from Breeze Graph Grammar, xBreeze/ADL uses the Rule to specify the style constraint and user-defined constraint. More details about how to use Rule to define those constraints will be explained in Section 5.1.

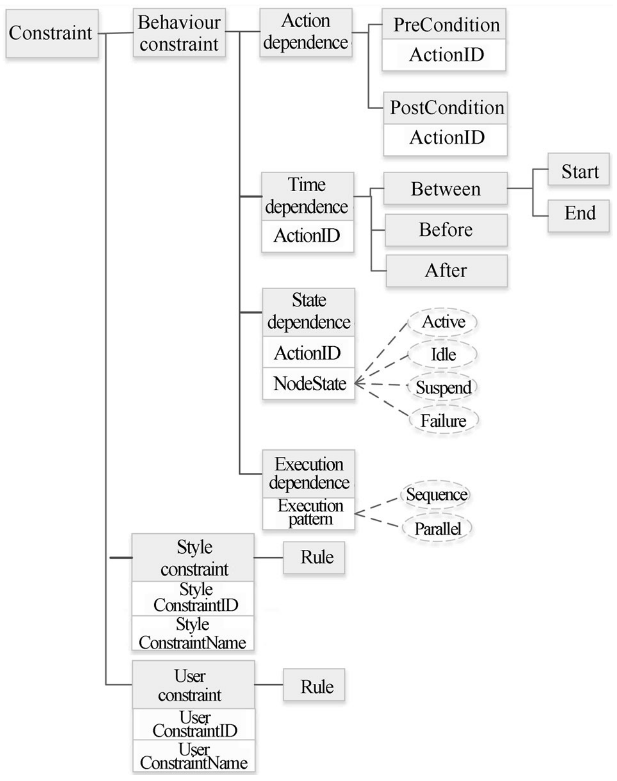


Fig. 7 Constraint schema: defining behaviour, style and user constraint

4.6 Instance schema

Based on the architecture templates, styles and constraints, an architecture instance can be generated. Each instance has its own template which provides a basic structure and semantic information. The composition of architecture instances needs to follow the style constraints. Though the instances are created by templates, xBreeze/ADL also supports incremental modification, i.e., developers may add some other attributes, actions, ports and connections only if these modifications are not violating the original behaviours and constraints which are inherited from the templates.

xBreeze/ADL describes some key features of an architecture instance, i.e., Node Instance, Super Node Instance, Edge, Style. Fig. 8 shows the corresponding meta elements of the Instance schema. Unlike the template, an architecture instance needs to declare its style when it is generated. Since an architecture instance may have several sub-systems, the StyleID attribute helps to identify which style the sub instance refers to. Each node instance and super node instance have to connect to their templates through NodeTemplateID attribute and SNodeTemplateID attribute respectively. If the developer adds a new port to the node instance without violating constraints, the semantic and constraints of the new port need to be defined. Otherwise, the PortTemplateID attribute of the instance port needs to refer to its port template. The instances generated from the templates do not need to specify their semantics or constraints again. The Edge element helps to build the interaction among the components.

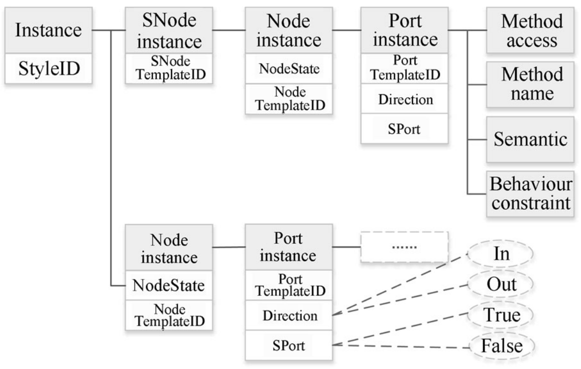


Fig. 8 Instance schema: defining an architecture instance according to templates

5 Architecture configuration management schemas

Software systems have always been in a changing, complex and uncontrollable environment which brings new challenges to the software architecture configuration management. Most of proposed ADLs are either used to capture the static structure information of the software system at architecture level, or using formal notations to represent the architecture evolution in a complex way.

In order to specify possible changes during the system running time and verify the architecture constraints, xBreeze/ADL uses the idea of graph transformation and defines Production element to implement style constraints and user-defined constraints definition and verification. It also specifies essential reconfiguration operations for the architecture evolution. Besides, xBreeze/ADL not only supports basic reconfiguration operations, including Addition, Removal and Replacement operations, but also is able to specify the operation pattern of those operations, e.g., sequence, parallel and iteration.

5.1 Rule schema

Before continuing, it is necessary to have a short discussion on the definition of production.

In graph grammar, a production is also called a graph rewriting rule. Generally, a production has two graphs, i.e., left-hand side (LHS) graph (pre-condition) and right-hand side (RHS) graph (post-condition). The graph transformation is implemented by using production. In Breeze graph grammar, productions not only are over the same alphabets of node and edge labels, i.e., L_N and L_E, but also rely on their ports and nodes attributes, i.e., A_{CN} and A_N. A production (graph rewriting rule) can be written as $P = (L, R)$, i.e., a pair of graphs (i.e., LHS and RHS graph) over the same alphabets of node and edge labels.

In order to achieve the software architecture reconfiguration (graph transformation), we need to find a redex, a matching sub-graph in the host graph, and use the RHS to replace the LHS in this redex. This match processing is also looked as a mapping or morphism between the LHS and RHS graph. More details can be found in [1, 2].

xBreeze/ADL capitalizes the above graph transformation and defines the Production element in Rule schema. It uses the LHS and RHS elements to represent the corresponding left-hand side graph and right-hand side graph of a production in Breeze graph grammar, respectively. The LHS defines the precondition of a reconfiguration operation including the related structure and semantic. The RHS describes the postcondition (i.e., new structure and (or) semantic) after the operation executions. The constraints, including styles and user-defined constraints, can be also defined and verified through the Production. Unlike the reconfiguration operations, the LHS of a constraint represents the architecture structure and (or) semantic which violate constraints, and the RHS refers to the correct situation. For architecture constraints verification, if the LHS of the constraints is found in the architecture instance, then we can infer that the architecture does not satisfy the constraints. The user-defined constraint can also be performed through this way. The next section will explain how to use the Rule to implement the reconfiguration operations. Fig. 9 is the corresponding meta elements of the Rule schema.

5.2 Configuration schema

In order to capture the change during both the initial development and subsequent evolution, xBreeze/ADL uses the Configuration schema to define three reconfiguration operations (i.e., addition, removal and replacement) for the architecture evolution, and also considers composite operations (i.e., sequence, parallel and iteration).

Configuration schema declares two elements: Reconfiguration and RunningPattern. Reconfiguration defines three subelements: Addition, Removal and Replacement. Each subelement owns a Rule element which has a set of corresponding productions. RunningPattern arranges the above productions in the way of sequence, parallel or iteration according to the system requirements. Fig. 10 shows the meta elements of the configuration schema.

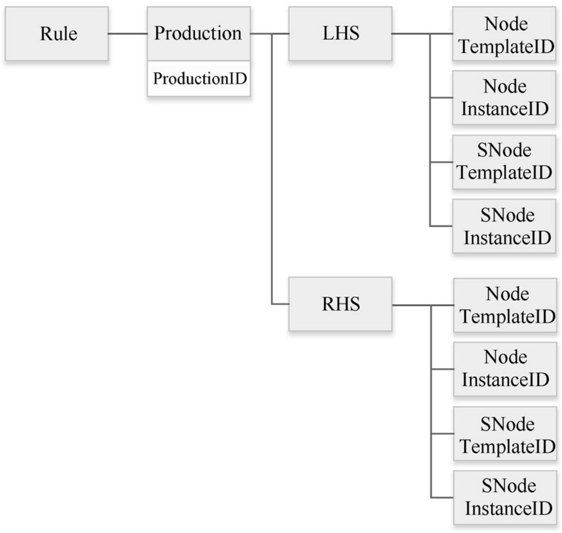


Fig. 9 Rule schema: defining rules for style constraint and reconfiguration operation

6 Case study

In this section, we use an online movie ticket booking system (OMTBS) to explain how to model software architecture with xBreeze/ADL. The graphic notation of online booking system is described in Fig. 11.

The OMTBS helps people to book the movie ticket. Customers may use three types of Client, i.e., Cell phone, Laptop and PAD, to book the movie tickets through a Booking agent. All the booking information collected by Booking agent are registered at Booking server. The Booking agent will receive the results from the Booking server as long as the process is completed.

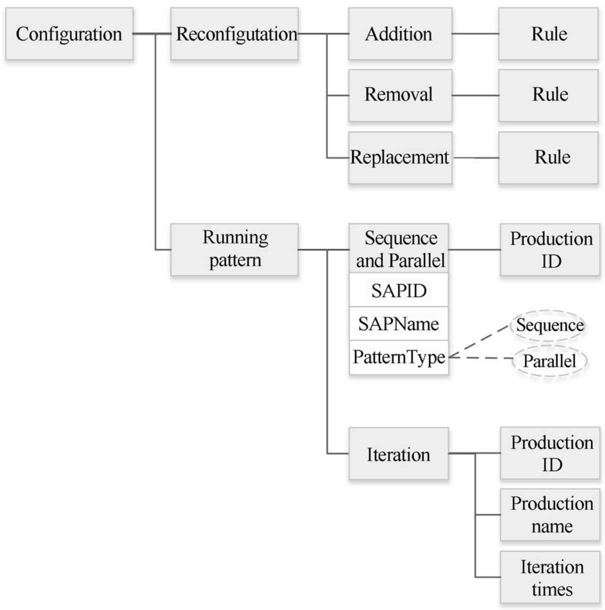


Fig. 10 Configuration schema: defining the reconfiguration and running pattern

6.1 Architecture modelling for OMTBS

In order to model the software architecture of OMTBS, we need to specify the architecture template, style, constraint and instance first. Due to the page limit, we only provide essential parts of the modelling results.

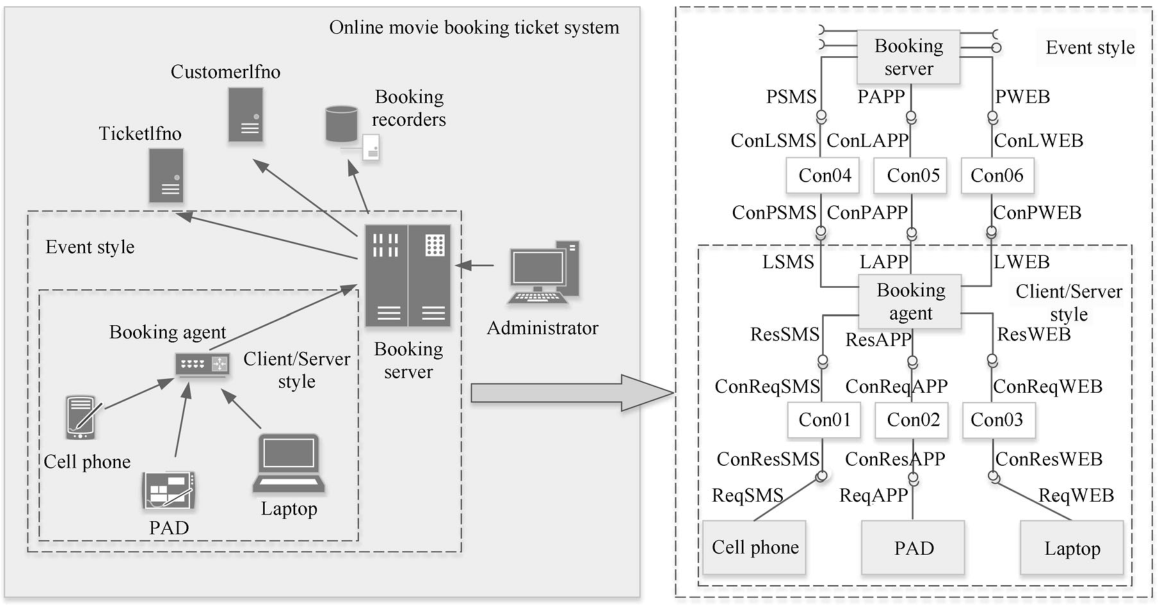


Fig. 11 Online movie ticket booking system

6.1.1 Template for OMTBS

We take the Booking server and Booking agent as examples to show how to define the template for OMTBS.

Fig. 12 presents a node template for the Booking server, and it specifies the Type and Class of Booking server as Component and BookingServer respectively.

```
<NodeTemplate NTID=NTBS Type=Component Class=BookingServer>
 <PortTemplate PTID=PTPSMS  Direction=out SPort=false>
  <MethodAccess>Public</MethodAccess>
  <MethodName>BookingProcess</MethodName>
  ......
 </PortTemplate>
  ......
</NodeTemplate>
<SNodeTemplate SNTID=SNTBC name=BookingClient>
 <NodeTemplate NTID=NTBA Type=Component Class=BookingAgent>
  <PortTemplate PTID=PTLSMS Direction=in SPort=true>
   <MethodAccess>Private</MethodAccess>
   <MethodName>BookingTicket</MethodName>
   ......
  </PortTemplate>
  ......
 </NodeTemplate>
  ......
</SNodeTemplate>
```

Fig. 12 Templates for booking server and booking agent

One of the port templates−PTPSMS of the Booking server is defined as a provider port (i.e., direction is out) to answer the booking requests from Booking client. As a provider port, its method access type is required to be Public for other component invocations. Its method name is defined as the BookingProcess. Since it is a regular port template, the SPort attribute is set to false. The Booking agent transfers the booking requests from Cell phone, PAD and Laptop to the Booking server in Fig. 11. As a super node−BookingClient is defined, which contains the Booking agent, Cell phone, PAD and Laptop templates, and shown as a client in Fig. 11. The super node capitalizes the port (PTLSMS) of Booking agent to communicate with the Booking server, and the SPort attribute of PTLSMS is set to true.

6.1.2 Semantic for OMTBS

The semantic of the Booking Server is depicted by its behaviour, i.e., answering the booking requests. Fig. 13 presents the corresponding semantic which separates the behaviour into two actions – CalleeBStSMS and ResultBStSMS. The service provider port template (PTPSMS) of Booking server uses its Provider element to model the behaviour of booking ticket through SMS. It expects the String and Date type of the parameters from the requestor and then returns a String type result.

6.1.3 Links for OMTBS

The link is represented by the edge in xBreeze/ADL. It stands for the connection for the component communication. In Fig. 14, the first Edge (ConPSMS) defines the link between the Booking agent (NTBS) and Connector (Con04, i.e., connector template). Since the request is sent from the Con04 to the NTBS, the ConPSMS sets the Con04 as a source node and connects its port (PTConLSMS) to the target port (CalleeBStSMS) of the NTBS. It also binds their actions together to refine the connection among the components.

```
<NodeTemplate NTID=NTBS Type=Component Class=BookingServer>
 ......
<PortTemplate PTID=PTPSMS Direction=out SPort=false>
  ......
  <Semantic>
    <Provider ProviderID=PIDBStSMS>
     <Callee ActionID=CalleeBStSMS>
      <MethodParameter>String</MethodParameter>
      <MethodParameter>Date</MethodParameter>
    </Callee>
    <Result ActionID=ResultBStSMS>
     <MethodReturn>String</MethodReturn>
    </Result>
  </Semantic>
 </PortTemplate>
 ......
</NodeTemplate>
```

Fig. 13 Semantic for booking server

One of the advantages of xBreeze/ADL is providing virtual edge and concrete edge to define the connection among components and within a component respectively. Thus, the first and second edges of Fig. 14 are between the component and connector, they are concrete edges and their Virtuality are both false. The third edge shows that the Booking agent uses its port (PTResSMS) to receive booking requests from the Cell phone, and delivers the requests to the Booking server through its port (PTLSMS).

```
......
<Edge EdgeID=ConPSMS Virtuality=false >
 <Source SourceNodeID=Con04 SourcePortID=PTConLSMS
   RequestID=RIDCon04 ActionID=CallerCon04 />
 <Target TargetNodeID=NTBS TargetPortID=PTPSMS
   ProviderID=PIDBStSMS ActionID=CalleeBStSMS/>
</Edge>
<Edge EdgeID=LConSMS Virtuality=false >
 <Source SourceNodeID=SNTBC SourcePortID=PTLSMS
   RequestID=RIDSMStBs ActionID=CallerSMStBS />
 <Target TargetNodeID=Con04 TargetPortID=PTConPSMS
   ProviderID=PIDCon04 ActionID=CalleeCon04/>
</Edge>
<Edge EdgeID=ResLSMS Virtuality=true >
  <Source SourceNodeID=NTBA SourcePortID=PTResSMS />
  <Target TargetNodeID=NTBA TargetPortID=PTLSMS />
</Edge>
 ......
```

Fig. 14 Links for Booking server and Booking agent

6.1.4 Style for OMTBS

Most of the large, complex software systems might contain many subsystems and each of them may follow different styles. In our example, the Booking Client submits the ticket booking information to the Booking server, and the Booking server will send events to the Booking client to awake the Booking agent to deliver the booking results to the customers. This is a typical Event style. Besides, the super node – Booking client follows the client/server style.

Three essential elements need to be specified in the architecture style, i.e., template, edge and style constraints. Fig. 15 shows part of the event style definition of Booking server and Booking client. The event style requires two types of node templates – event publisher (NTBS) and event listener (SNTBC). The edges between the NTBS and SNTBC are also defined. For example, the edge ConPSMS and LConSMS help the communication of processing the booking request through SMS. The StyleConstraintID references the corresponding style constraints which are defined in the Constraints part.

6.1.5 Constraints for OMTBS

xBreeze/ADL encapsulates three types of the constraints, i.e., behaviour constraints, style constraints and user-defined constraints.

```
<Architecture>
 ……
<Style StyleID=Style01 StyleName=Event>
  <NodeTemplate NodeTemplateID= NTBS>
    <PortTemplateID>PTPSMS</PortTemplateID>
    ……
  </NodeTemplate>
   <NodeTemplate NodeTemplateID= SNTBC>
    <PortTemplateID>PTLSMS</PortTemplateID>
    ……
   </NodeTemplate>
   <NodeTemplate NodeTemplateID= NTCon04>
    <PortTemplateID>PTConPSMS</PortTemplateID>
    <PortTemplateID>PTConLSMS</PortTemplateID>
   </NodeTemplate>
    ……
   <Edge EdgeID=ConPSMS Virtuality=false >
     <Source SourceNodeID=Con04 SourcePortID=PTConLSMS />
     <Target TargetNodeID=NTBS TargetPortID=PTPSMS />
   </Edge>
   <Edge EdgeID=LConSMS Virtuality=false >
     <Source SourceNodeID=SNTBC SourcePortID=PTLSMS  />
     <Target TargetNodeID=Con04 TargetPortID=PTConPSMS />
   </Edge>
    ……
   <StyleConstraintID>Event01</StyleConstraintID>
 </Style>
       ……
</Architecture>
```

Fig. 15 Event style definition

Fig. 16 gives the part of behaviour constraints for Booking Client and Booking server. The ActionDependence specifies the action (CallerSMStBS) of the port template (PTPSMS) of Booking client (SNTBC) that should happen before the action (CallerSMStBS) of the port template (PTLSMS) of Booking server (NTBS). The TimeDependence requires that the action (CallerSMStBS) of port template (PTLSMS) of Booking server (NTBS) is only allowed to be invoked between 9:00 to 22:00 per day. Besides, the state of Booking server (NTBS) should be active when the action (CallerSMStBS) starts. Three actions, i.e., CallerSMStBS, CallerAPPtBS and CallerWEBtBS, of Booking Agent (NTBA) could be executed in a parallel way.

In OMTBS, the Booking client, containing four nodes−Booking agent, Cell phone, PAD and Laptop, is a super node. It can be looked as a subsystem which follows the Client/Server style. One of its style constraints – NotClienttoClient01 is defined through the Production. In the super node (Booking client), the Booking agent is the server and the rest of the components (i.e., Cell phone, PAD and Laptop) are clients. Thus, all communications happen between the server and clients, and the clients cannot interact with each other. The LHS of the Production describes that the node template of Cell phone (NTCP) connects with the node template of PAD (NTPAD) through the connector template (NTCON). There are two edges (CPtCon and ContPAD) helping build connections between them, which violate the constraints of client/server style. The RHS removes the edges and eliminates this violation. The user-defined constraints can also be specified through the Productions.

6.1.6 Instance for OMTBS

Fig. 17 describes two instances of Booking Server and Booking Agent, i.e., NIBS (referring to the NTBS template) and NIBA (referring to the NTBA template). Their port instances are PIPSMS and PILSMS which come from the PTPSMS template and PTLSMS template separately. Two edge instances are also described, i.e., InsConPSMS and InsLConSMS. Since those ports are created by the templates which bind their basic actions together, the edges only need to connect the port instances together here. The StyleID attribute indicates that the instance follows the Event style which is already defined in the Constraints part.

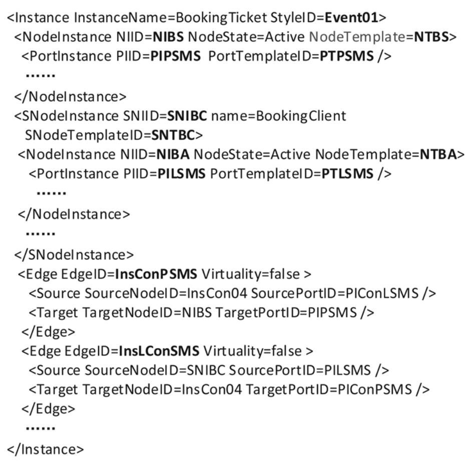
```
<Constraint>
 <BehaviorConstraint>
  <ActionDependence ActionDependenceID=ADIDSMStBS >
    <Precondition NodeID=SNTBC PortID=PTLSMS ActionID=CallerSMStBS/>
    <Postcondition NodeID=NTBS PortID= PTPSMS ActionID=CalleeBStSMS/>
  </ActionDependence>
 <TimeDependence TimeDependenceID=TDIDBStSMS NodeID=NTBS PortID=PTPSMS
ActionID=CalleeBStSMS >
  <Between>
    <Start>9:00</Start>
    <End>22:00</End>
  </Between>
 </TimeDependence>
 <StateDependence StateDependenceID=SDIDBStSMS  NodeID=NTBS PortID=PTPSMS
ActionID=CalleeBStBC  NodeState=Active />
 <ExecutionDependence ExecutionDependenceID=EDIDBCL
ExecutionPattern=Parallel >
  <Action  NodeID=NTBA PortID=PTLSMS ActionID=CallerSMStBS />
  <Action  NodeID=NTBA PortID=PTLAPP ActionID=CallerAPPtBS />
  <Action  NodeID=NTBA PortID=PTLWEB ActionID=CallerWEBtBS />
 </ExecutionDependence>
    ……
  </BehaviorConstraint>
 <StyleConstraint StyleConstraintID=Client/Server01 >
 <Rule>
  <Production ProductionID=NotClienttoClient01>
    <Description>Cell Phone cannot connect to the PAD</Description>
    <LHS>
     <NTID>NTCP</NTID>
     <NTID>NTPAD</NTID>
     <NTID>NTCon</NTID>
     <Edge EdgeID=CPtCon Virtuality=false>
       <Source SourceNodeID=NTCP SourcePortID=PTReqSMS />
       <Target TargetNodeID=NTCon TargetPortID=PTRes />
     </Edge>
     <Edge EdgeID=ContPAD Virtuality=false>
       <Source SourceNodeID=NTCon SourcePortID=PTReq />
       <Target TargetNodeID=NTPAD TargetPortID=PTResAPP />
     </Edge>
    </LHS>
    <RHS>
      <NTID>NTCP</NTID>
      <NTID>NTPAD</NTID>
      <NTID>NTCon</NTID>
    </RHS>
   </Production>
   ……
  </Rule>
</StyleConstraint>
 <UserConstraint>……</UserConstraint>
</Constraint>
```

Fig. 16 Specification of constraints

6.2 Architecture configuration management for OMTBS

In order to support the architecture dynamic evolution, xBreeze/ADL supports three types of reconfiguration operations (i.e., addition, removal and replacement). It also assigns the running pattern (i.e., sequence, parallel and iteration) to the above reconfiguration operations for describing the composite operations.

Fig. 18 gives two Addition operations, adding an instance of Cell phone (NICP02) and adding an instance of PAD (NIPAD04), to the OMTBS. Two productions, AddCell-Phone02 and AddPAD04, help to build the connections between the new instances and the two connectors (i.e., NICon01 and NICon02), respectively. The SequenceAnd-Parallel indicates such two productions can be executed in parallel.



```
<Instance InstanceName=BookingTicket StyleID=Event01>
 <NodeInstance NIID=NIBS NodeState=Active NodeTemplate=NTBS>
  <PortInstance PIID=PIPSMS  PortTemplateID=PTPSMS />
  ......
 </NodeInstance>
 <SNodeInstance SNIID=SNIBC name=BookingClient
   SNodeTemplateID=SNTBC>
  <NodeInstance NIID=NIBA NodeState=Active NodeTemplate=NTBA>
   <PortInstance PIID=PILSMS PortTemplateID=PTLSMS />
    ......
  </NodeInstance>
   ......
 </SNodeInstance>
  <Edge EdgeID=InsConPSMS Virtuality=false >
   <Source SourceNodeID=InsCon04 SourcePortID=PIConLSMS />
   <Target TargetNodeID=NIBS TargetPortID=PIPSMS />
  </Edge>
  <Edge EdgeID=InsLConSMS Virtuality=false >
   <Source SourceNodeID=SNIBC SourcePortID=PILSMS />
   <Target TargetNodeID=InsCon04 TargetPortID=PIConPSMS />
  </Edge>
  ......
</Instance>
```

Fig. 17 Instance of booking server and booking agent

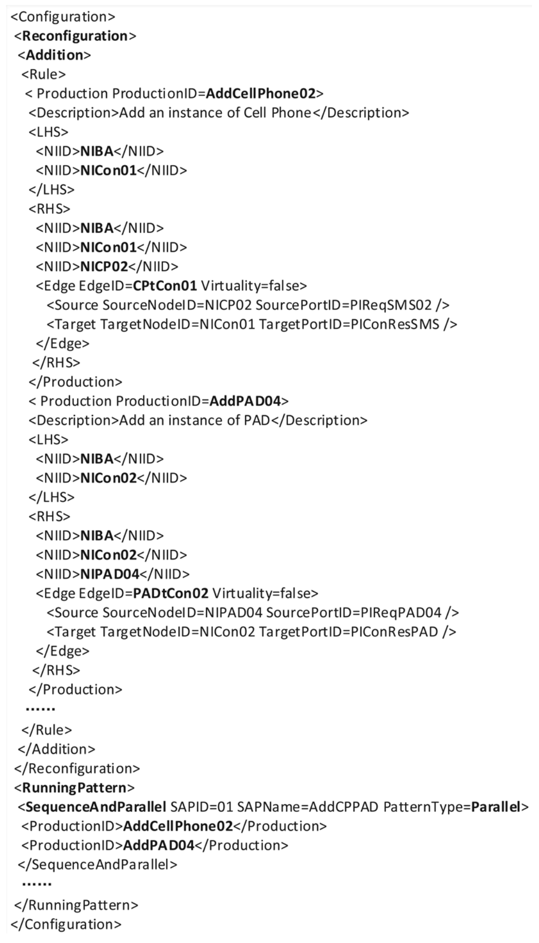
6.2.1 Summary

In this section, a double-style (i.e., Event style and client/server style) system−OMTBS is introduced to show how to model software architecture in xBreeze/ADL. By taking the advantages of other software architecture description techniques (e.g., xADL, AGG, rapide, Breeze/ADL), xBreeze/ADL can specify the fundamental structure information of OMTBS (e.g., templates and instance of booking server and booking agent), and the dynamic (e.g., constraint, reconfiguration) behaviour. Especially, the virtual/concrete link, reconfiguration operation, running pattern and the constraint representations which help to understand how the components interact with each other in OMTBS according to the system or customer′s requirements.

7 Conclusions and future work

In this paper, we proposed an ADL−xBreeze/ADL to specify and analyse the software architecture for large and complex software systems.

The advanced features of xBreeze/ADL are as follows: 1) it provides a flexible specification by leveraging the XML as its meta language, 2) it enhances the extensibility of architecture by separating the template from instance, 3) it identifies service execution flows within a component by using the virtual link, 4) it depicts the semantics by specifying the component behaviours, 5) it deals with constraints issues by capitalizing the multiple restrictions, such as the action dependence, time dependence, state dependence and execution dependence, and 6) it implements the architecture verification and reconfiguration for enhancing the architecture dynamism and evolvability by introducing the graph transformation.



```
<Configuration>
 <Reconfiguration>
  <Addition>
   <Rule>
    < Production ProductionID=AddCellPhone02>
    <Description>Add an instance of Cell Phone</Description>
    <LHS>
     <NIID>NIBA</NIID>
     <NIID>NICon01</NIID>
    </LHS>
    <RHS>
     <NIID>NIBA</NIID>
     <NIID>NICon01</NIID>
     <NIID>NICP02</NIID>
     <Edge EdgeID=CPtCon01 Virtuality=false>
      <Source SourceNodeID=NICP02 SourcePortID=PIReqSMS02 />
      <Target TargetNodeID=NICon01 TargetPortID=PIConResSMS />
     </Edge>
    </RHS>
    </Production>
    < Production ProductionID=AddPAD04>
    <Description>Add an instance of PAD</Description>
    <LHS>
     <NIID>NIBA</NIID>
     <NIID>NICon02</NIID>
    </LHS>
    <RHS>
     <NIID>NIBA</NIID>
     <NIID>NICon02</NIID>
     <NIID>NIPAD04</NIID>
     <Edge EdgeID=PADtCon02 Virtuality=false>
      <Source SourceNodeID=NIPAD04 SourcePortID=PIReqPAD04 />
      <Target TargetNodeID=NICon02 TargetPortID=PIConResPAD />
     </Edge>
    </RHS>
    </Production>
   ......
   </Rule>
  </Addition>
 </Reconfiguration>
 <RunningPattern>
  <SequenceAndParallel SAPID=01 SAPName=AddCPPAD PatternType=Parallel>
   <ProductionID>AddCellPhone02</Production>
   <ProductionID>AddPAD04</Production>
  </SequenceAndParallel>
   ......
 </RunningPattern>
</Configuration>
```

Fig. 18 Addition operations and the parallel execution

Currently, we are working on constructing an xBreeze/ADL toolset to model the software architecture on both static and dynamic aspects based on Breeze toolset, since the Breeze already supports the basic constraint and reconfiguration definition and execution. The next step of xBreeze/ADL development is to define template and instance separately, support more style definitions, implement semantic expressing and achieve the virtual/concrete link representation.

References

[1] C. Li, L. P. Huang, L. X. Chen, C. Y. Yu. Breeze/ADL: Graph grammar support for an XML-based software architecture description language. In *Proceedings of the 37th IEEE Computer Software and Applications Conference*, IEEE, Kyoto, Japan, pp. 800–805, 2013.

[2] C. Li, L. P. Huang, L. X. Chen, C. Y. Yu. BGG: A graph grammar approach for software architecture verification

and reconfiguration. In *Proceedings of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IEEE, Taichung, Taiwan, China, pp. 291–298, 2013.

[3] R. Allen, D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 3, pp. 213–249, 1997.

[4] R. Allen, R. Douence, D. Garlan. Specifying and analyzing dynamic software architectures. In *Proceedings of the First International Conference on Fundamental Approaches to Software Engineering*, Springer, Lisbon, Portugal, vol. 1382, pp. 21–37, 1998.

[5] D. Garlan. *An Introduction to the Aesop System*, [Online], Available: http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/html/aesopoverview.ps, May 10, 2015.

[6] A. Terry, R. London, G. Papanagopoulos, M. Devito. The ARDEC/Teknowledge Architecture Description Language (ArTek), Version 4.0, Technical Report, Teknowledge Federal System and U. S. Army Armament Research, Development and Engneer. Centery, USA, 1995.

[7] N. Medvidovic, P. Oreizy, J. E. Robbins, R. N. Taylor. Using object-oriented typing to support architectural design in the C2 style. *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 6, pp. 24–32, 1996.

[8] P. Binns, M. Englehart, M. Jackson, S. Vestal. Domain-specific software architectures for guidance, navigation and control. *International Journal of Software Engineering and Knowledge Engineering*, vol. 6, no. 2, pp. 201–227, 1996.

[9] W. Tracz. LILEANNA: A parameterized programming language. In *Proceedings of IEEE 2nd International Workshop on Software Reusability*, IEEE, Lucca, Italy, pp. 66–78, 1993.

[10] M. Moriconi, R. A. Riemenschneider. Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies, Technical Report SRI-CSL-97-01, SRI International, USA, 1997.

[11] M. M. Gorlick, R. R. Razouk. Using weaves for software construction and analysis. In *Proceedings of IEEE 13th International Conference on Software Engineering*, IEEE, Austin, USA, pp. 23–34, 1991.

[12] J. Magee, N. Dulay, S. Eisenbach, J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, Springer, Sitges, Spain, vol. 989, pp. 137–153, 1995.

[13] D. C. Luckham, J. Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, vol. 21, no. 9, pp. 717–734, 1995.

[14] M. Shaw, R. DeLine, D. Klein, T. Ross, D. Young, G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 314–335, 1995.

[15] D. Garlan, R. Monroe, D. Wile. ACME: An architecture description interchange language. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, ACM, Ontario, Canada, pp. 159–173, 1997.

[16] E. M. Dashofy, A. Van der Hoek, R. N. Taylor. A highly-extensible, XML-based architecture description language. In *Proceedings of 2001 Working IEEE/IFIP Conference on Software Architecture*, IEEE, Amsterdam, The Netherlands, pp. 103–112, 2001.

[17] H. Mei, F. Chen, Q. X. Wang, Y. D. Feng. ABC/ADL: An ADL supporting component composition. In *Proceedings of the 4th International Conference on Formal Engineering Methods*, Springer, Shanghai, China, vol. 2495, pp. 38–47, 2002.

[18] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[19] G. Taentzer. AGG: A graph transformation environment for modeling and validation of software. In *Proceedings of the 2nd International Workshop, Lecture Notes in Computer Science*, Springer, Charlottesville, USA, vol. 3062, pp. 446–453, 2004.

[20] W. P. Jiao, H. Mei. Automated adaptations to dynamic software architectures by using autonomous agents. *Engineering Applications of Artificial Intelligence*, vol. 17, no. 7, pp. 749–770, 2004.

Chen Li received the B. Sc. degree in computer science and technology from University of Science and Technology of China, China in 2003, the M. Sc. degree in computer applications technology from the University of Shanghai for Science and Technology, China in 2010, and the Ph. D. degree in computer science and technology from Shanghai Jiao Tong University, China in 2015. Currently, he is a postdoctoral research assistant in School of Humanities and Cultural Industries at Bath Spa University, UK. He has published about 28 refereed journal and conference papers. He is a member of CCF and IEEE.

His research interests include software architecture, software reliability and formal methods.

E-mail: c.li2@bathspa.ac.uk

ORCID iD: 0000-0001-6249-8957

Hong-Yi Yang received the B. Sc. and M. Sc. degrees in computer science from the Jilin University, China in 1982 and 1985, respectively, and the Ph. D. degree in computer science from Durham University, UK in 1994. He was a faculty member at Jilin University, China in 1985, at Durham University, UK in 1989, at De Montfort University, UK in 1993, and at Bath Spa University, UK in 2013. Currently, he is a professor in School of Humanities and Cultural Industries at Bath Spa University, UK. He has published about 400 refereed journal and conference papers. He has become IEEE Computer Society Golden Core member since 2010. Also, he is a member of EPSRC Peer Review College

since 2003. He is the editor in chief of International Journal of Creative Computing, InderScience.

His research interests include software engineering, creative computing, web and distributed computing.

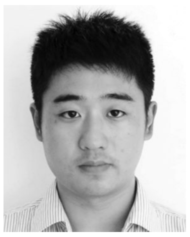
E-mail: h.yang@bathspa.ac.uk (Corresponding author)

ORCID iD: 0000-0001-6561-3631

Mei-Yu Shi received the B. Sc. and M. Sc. degrees in economic from the Jilin University, China in 1990 and 1997, respectively, and the Ph. D. degree in economic from Graduate School of Chinese Academy of Social Sciences, China in 2003. Currently, she is a professor at the Tourism Institute of Beijing Union University, China. She has published about 40 refereed journal and conference papers.

Her research interests include intangible cultural heritage tourism development, tourism shopping and tourism products development, convention and exhibition tourism, tourism destination marketing.

E-mail: shimeiyu72@163.com



Wei Zhu received the Ph. D. degree in computer science and computer engineering from La Trobe University, Australia in 2013. Currently, he is a lecturer of Tourism Institute of Beijing Union University, China.

His research interests include artificial intelligence and data mining in tourism.

E-mail: wzhu83@163.com