2012-09-22

PrimitiveC-ADL: Primitive Component Architecture Description Language

Basel Magableh
Dublin Institute of Technology, 453543@dit.ie

PrimitiveC-ADL: Primitive Component Architecture Description Language

Basel Magableh
Department of Computer Science and Statistics
Trinity College of Dublin
Email:magablb@cs.tcd.ie

Stephen Barrett
Department of Computer Science and Statistics
Trinity College of Dublin
Email:stephen.barrett@tcd.ie

***Abstract*—In this paper, we introduce an architecture description language (ADL) for PCOMs (a context oriented component model). The language is described at three levels: (1) Building blocks (PCOMs context oriented components types) (2) Connectors, which connect components externally and internally, and (3) Architectural Configuration, which includes a full description of composition and decomposition mechanisms.**

The contribution is designing ADL. That supports context-orinted component by providing new architecture elements, which fulfil the requirements of designing context oriented component based applications. Context oriented component is a behavioural unit composed of static parts and dynamic parts. A PCOMs component model design was introduced in our previous work. PCOMs proposes a component model design to compose context-aware system by capturing context condition at runtime. The model is a component-based one that modifies the application architecture by subdividing components into subsystems of static and dynamic elements. We map each context condition to a composable template architectural configuration. Each context condition acts to select behavioural patterns, which combine to form application architectures.

Different types of architecture elements are proposed in this work. We focus in defining the following new elements: Components' dynamic and static parts, components' layers, decision policies, and composition plan. Finally we introduce an ADL that fully supports context aware applications, by supporting the definition of a component as a unit of behaviour. Our ADL clearly defines the composition mechanisms, and provides proper definition for the composition's design Patterns and composition plan.

A Context oriented component is a behavioural unit composed with static parts and dynamic parts. A PCOMs component model design was introduced in our previous work. PCOMs proposes a component model design to compose context-aware system by capturing context condition at runtime. The model is a component-based one that modifies the application architecture by subdividing components into subsystems of static and dynamic elements. We map each context condition to a composable template architectural configuration. Each context condition acts to selected behavioural patterns, which combine to form application architectures.

***Index Terms*—Context Oriented Component Model; Architecture Description Language; Composition Definition Language; PrimitiveC-ADL; Context Orinted ADL;**

I. Introduction

Architectural design has always played a strong role in determining the success of complex software-based systems. Choosing an appropriate architecture description language can allow the production of software that satisfies its requirements and is easily modified as new requirement present. Several researchers have attempted to surveying what they consider ADLs or by listing essential requirement for ADL. This paper builds upon the result of previous researcher investigation.

Szyperski's definition of a specific software component is a unit of composition with contractually specified interfaces and explicit context dependencies only [1]. This standard component model does not support the composition of context-aware application. It does not deal with components as a behavioural units.

Generally, the software component model is a definition of the semantics, the syntax and the composition of component [2]. Components can be defined and constructed. The definition of components requires a component definition language which may be distinct from the implementation language.

A PCOMs component was proposed in our previous work [3]. The PCOMs component definition is different from the definition of component in aspect oriented or in object oriented. PCOMs proposes a component model design to compose context-aware system by capturing context condition. The model is a component-based one that modifies the application architecture by subdividing components into subsystems of static and dynamic elements. We map each context condition to a composable template architectural configuration. Each context condition acts to select behavioural patterns, which combine to form application architectures.

ADLs are formal languages used to represent the architecture of software system. Components, behavioural specifications, patterns and interaction mechanism all together described in a comparative study [4]. The current ADLs does not support context oriented component types. Context oriented components have specific requirements that should be described by the ADL. PCOMs have many elements such as the component's dynamic and static parts, layers, decision points and decision policies. These elements should be defined in the architecture description language.
The Architecture definition language should clearly support context oriented component by fulfilling the following requirements:

1) Defining a component as units of behaviour. This mechanism allows components to dynamically modify their functional behaviour at runtime.
2) Defining the component's dynamic parts, that represent

the component unanticipated behaviour.
3) Defining the component static parts, that represents the context-unsenstive elements.
4) Defining the application composition mechanism, by supporting the definition of internal and external component composition.
5) Defining the component's layers and decision points.
6) Describing the component's ports and interfaces.
7) Support architectural configuration by clearly describing the components' hierarchical dependent graph and the components' interaction.
8) Supporting the description of application composition plan for external composition type, and design pattern for internal composition.
9) Differentiate between the type of connectors used in external composition and the connectors used in the internal composition.

There is no existing ADL that we are aware of, which explicitly defines context aware components as architectural entity.

II. Related work

Architecture description language contains the following elements: (1) Components, representing the primary computational elements of a system. (2) Connectors, representing interactions between components, manipulating the communication and the coordination activities between components. (3) Systems, representing the configuration graphs of component and connectors. (4) Proprieties, representing semantic information about a system and its components beyond the structure. (5) An architectural style, which defines design elements types and rules for component composition.

Most ADLs specified in literature are loosely coupled to implementation languages, That causes a problem in analysing, implementing, understanding and evolution software systems.

We closely examined a survey of several ADLs conducted by Taylor et. al. [4]. Another survey was conducted by Clements et. al. [5]. The authors in [6] proposed a runtime software reconfiguration by replacing single components at architecture level. TADL for trustworthy component based systems was introduced by [7]. XADL [8] and [9] propose an ADL for dynamic components and aspect oriented languages.

The comparative study for the above ADLs and ADL tools reveals the fact that the proposed ADLs are not designed or prepared to support context oriented component type.

In the Koala model, components are reusable units of design and development that support late binding. KOALA provide interface and configuration definition to support the components [10]. There is no port, layers or support for defining the component as unit of behaviour in KOALA.

Acme ADL [11] describes component composition by explicitly specifying component communication. Acme supports the use of multiple component interfaces. They support hierarchical description and encapsulation of components. Despite they supporting non-functional properties and providing explicit description of semantics communication infrastructure, but Acme does not include architecture elements that could manipulate the internal structure of a component.

Acme defined architectural structure using seven core entities: components, connectors, systems, ports, rule, representation and rep-map.Acme specify consistence definition of these elements with the normal definition that ADL defined.

The term port is introduced to identify a point of interaction between components and the environment. ADL, specify interfaces where Acme uses the port notation to represent an interface. Acme connectors not only describe the interaction among component, but they specify interfaces that are defined by a set of roles. The role notation used to define a participant of the interaction represented by the connector for example the sender and receiver roles of message passing connector.

Acme defines graphs in which the components are nodes and connectors are arcs. The graph representation allows the definition of which component port is connected to which connector role. Acme use representation maps to illustrate associations between internal ports and external ports, that support a hierarchal representation of the architecture. Acme uses properties to outline the architecture documentation. Constrains are used to define the boundary of the architectural design at runtime.

In Acme components are described using object oriented notation. Each component can be represented by a class. component interface represented by a class definition, and communication between components defined in term of associations.

The CDL (Composition Description Language) presented by Acme [11] covers some PCOMs requirements. Acme fails to provide application architecture configuration, It is obvious that Acme were not able to provide composition mechanisms. In order to define PCOMs component, Acme must be modified to include PCOMs core elements as: Layers, Decision Points, Decision policy and composition plan.

A Kobra component model was proposed in [12] as a UML stereotype with specification and implementation. The specification describes the component interface and what it does. The component implementation explains the mechanisem of how it does it. In Kobra no new composition of component is possible, so it fails to meet the requirement of PCOMs component model

In DAOP-ADL [13]. The authors attempt to provide a new ADL that specify the structure of a system in terms of components, aspects and composition rules. The composition rules define how to plug in components and when to apply aspects to components to extend the system behaviour. DAOP attempts to describe the ADL using XML and XML schema which support the integration and interpretation of the information specified by the ADL at runtime. The authors of DAOP claim this level of integration supports dynamic aspect weaving at runtime. DAOP fails to support PCOMs because it does not support changing subdivision of components' implementation by others to satisfy the current context.

DAOP-ADL [13] deals with components and aspects as first-order entities that are dynamically composed at runtime

by using dynamic weaving between components and aspects. DAOP deals with component as black-box, coarse-grained entities this restriction of granularity prevent any type of modification inside the component structure, which is the core concept specified by PCOMs components, as they are fine-grained component type, with dynamic and static parts.

In a classification study performed by [4]. Several ADLs were investigated. The study classifies and compares several ADLs in terms of architecture modelling features, architectural configuration and tool support.

The study contrasts between several ADLs: Aesop, ACME, arTek, C2, Darwin, LILEANNA, MetaH, Rapide, SADL, UniCon, Weaves and Wright. In Taylor's [4] study the author focuses in how the ADLs model the architecture configuration in term of understandability, compositionallity, Refinement, traceability, Heterogeneity, scalability, Evolution, Dynamism, constraints and Non-functional properties.

Medvidovic and Taylor [4] claim that C2 and weaves are the only ADLs which support unanticipated adaptation. Where the other ADLs fail or they partially support this feature. Based on that study we decided to go further to study C2 and weaves where they could support PCOMs component requirements.

The main features supported by C2 are it has explicit architectural topology, it allow internal composition via internal component architecture, it supports unanticipated dynamism via elements insertion, removal, and rewiring. But the interface point is provided by single ports which we think this is a weakness. Even ports in C2 are in different type to maximize the flexibility of interconnection. They model component behaviour via invariants and operation of pre and post conditions [14]. C2 is not suitable to be used to model PCOMs due to its weakness in providing clear architecture configuration and composition mechanism.

Taylor et. al. propose in [8], an XML-based architecture description language. XADL 2.0 and XArch tool are described which are claimed to be a second generation of ADL. The use of XML schema to support the basic elements of ADL was an advantage, which gives XADL an intensive extensible to express layers. Each ADL elements are defined as tags, so they can be extende to express more element's property. XML provides an ideal platform upon which to develop an extensible modeling language for software architectures.

Despite XADL has provides a definitions for component, connector , interface, links, and sub architectures. The definition of these elements are semantically neutral, as no behaviour is attached. This make defining context oriented component type a very hard task, becaues it is not possible to define components as units of behaviour by XADL.

Context Oriented Component ADL should at least supports the definition of some elements which are missing from the current state of art. From the above comparative study we conclude it necessary to design new ADL that supports context oriented component.

In our previous work [3], a component model design proposed to reach a high level of unanticipated dynamism adaptation in mobile devices. A middleware was proposed supported by a case study.

Unanticipated dynamism is the application ability to accommodate expected dynamic changes based on the application architecture configuration. In our designed model PCOMs, we showed a possibility to accomplish unanticipated adaptation by capturing the context at runtime, then mapping context condition to a composable template architectural configuration provided.

The architecture configuration leads the adjustment process by clearly defines: composition plan, design patterns, components, connectors, decision policies and the composition mechanism. The application's composition plan is more like building construction plan. It specifies the outline of the application, the main parts, and the connection between the application's components.

This what motivate us to design PrimitiveC-ADL, it describes components as unit of behaviour, component's sub architecture parts (the dynamic and static), component's layers, and it specify the composition mechanisms. Adding to the above. PrimitiveC-ADL is capable to provide a definition for the Design Patterns and a composition plan. In the following sections we are going to explain the component model first, then a full description of the new PrimitiveC-ADL we invent to describe the context oriented component.

III. PCOMs Component framework

We call any subpart of the software system that is associated with specific context condition the PCOMs component. A PCOMs component is a unit of behaviour contractually specified by interfaces and explicit dependencies (i.e. standard component model). A PCOMs component structure is shown in Figure 1. The PCOMs component has multiple ports and decision points.

Each port identifies a point of interaction between the component and its enviroment. A component may provide multiple interfaces by using different types of ports. A port can represent an interface as simple as a single procedure signature, or more complex interfaces, such as a collection of procedure calls that must be invoked in certain specified orders, or an event multi-cast interface point. The Decision Point identifies a point of interaction inside the component by activating or deactivating the component's layers.

PCOMs component consist of two parts: static behaviour (SB) and dynamic behaviour (DB) as in Figure **??**. The component's static behaviour part represents the application behaviour which is not context-dependent. The component dynamic behaviour part is context-dependent functionality directly affected by the execution context, such as location or bandwidth level. A component's dynamic parts have many layers. Each layers controls the activation or deactivation of more specific features associated with specific behaviour. The architecture description language (ADL) in Figure **??** shown the syntax definition of a PCOMs component with ports, dynamic parts and Layers.

Developers design one or more PCOMs components to map specific context information. For example the bandwidth

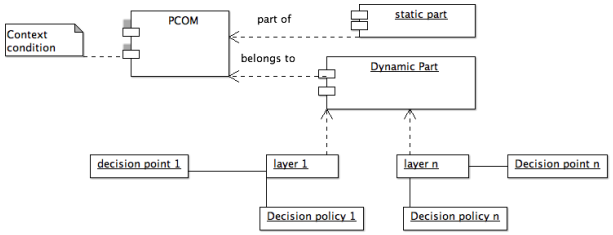

Fig. 1. PCOMs

status has three possible conditions (normal, low, or high). For each condition a PCOMs component is designed. For bandwidth PCOMs components zero or more decision points are embedded. Decision points are manipulated by decision policies at runtime. Decision policy is a predefined description used to control the PCOMs component's layers to manage the components' behaviours [15].

In the PCOMs framework, we use a *composition plan* and *design patterns* model. The *composition plan* recursively describes the composite components, and the connection between them by describing several *design patterns*. A single *composition plan* represents one possible realisation of the associated PCOMs component. A *design pattern* typically shows relationships and interactions between components' dynamic behaviour parts. It also describes the dependancies between components. Using composition plan and *design patterns* provides support for two type of composition. External composition and Internal composition.

A. Internal composition

Internal composition describes the composition inside the component by replacing its implementation with other components' subdivisions to satisfy specific dynamic behaviour.

The dependancies among components can be described using *design Patterns*. Components are bound together to satisfy their dependancies. The binding is performed at runtime and provides the necessary adjustments to introduce runtime behavioural variations. The specification of PCOMs components is driven by a context oriented selection process.

B. External composition

External composition illustrates the mechanism to connect components via ports and connectors. A Component is modelled as requiring and/or providing services from other components to achieve specific behaviour. A connector uses the description illustrated by the composition plan to connect components A and B. As in ACME [11].

The Connector consist of two parts: interfaces and layout. The interface is used as an input/output terminal between components. The layout describes the internal configuration of the connector. It shows the mechanism of connecting two interfaces internally.

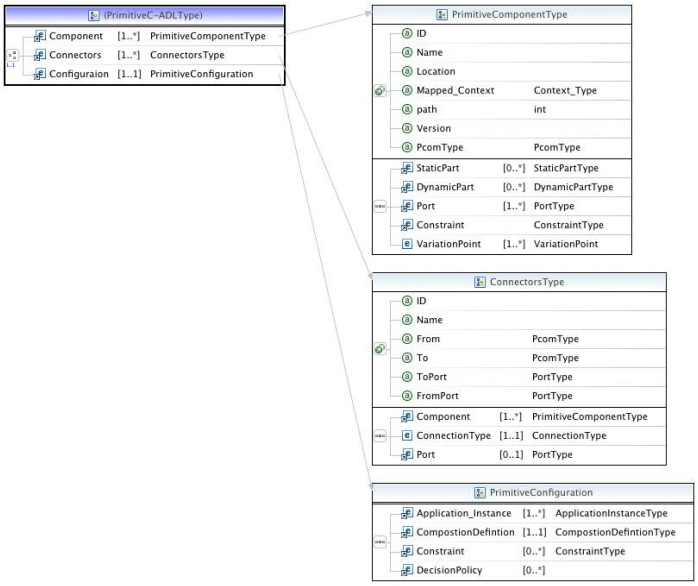

Fig. 2. PrimitiveC-ADL

IV. PrimitiveC-ADL: A Context Oriented Component Architecture Description Language

The contribution of this work is to introduce PrimitiveC-ADL which supports context-aware applications. PrimitiveC-ADL is associated with context information for building software system. In this research we are trying to bridge the gap between informal diagrams and programming language implementations towards building context-aware application from a well defined ADL.

We introduce PrimitiveC-ADL in terms of three levels (1) building blocks (PCOMs components) (2) Connectors, and (3) Architectural configuration that includes a full description of composition and decomposition mechanism. Figure 2 shows the main elements of PrimitiveC-ADL, Each element is associated with a architectural template type. The main features provide by element's type are instantiation, evolvtion and inheritance.

Each element is inherited from another architectural template, for example a component is inherited from PrimitiveComponentType as shown in Figure 2. The component element used to define the application components. The Connectors are used to connect components through the external composition. Configuration elements are used to describe the application's instance, the composition mechanism, the application's constraint, and the Decision policy.

A. Moddling the PrimitiveC-ADL Component

Tyalor et. al. [4] defines component as a unit of computation or a data store, the component could be small as a single procedure or large as an entire application. In our PCOMs component design, we deal with components model that

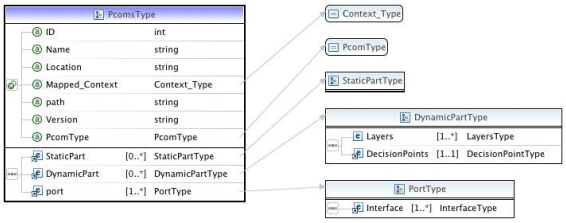

Fig. 3. Component Descriped by PrimitiveC-ADL

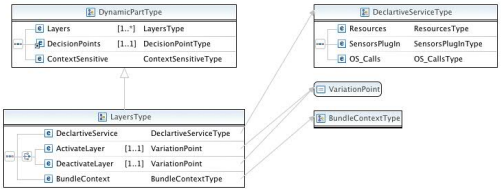

Fig. 4. A Component Dynamic Part in PrimitiveC-ADL

deploys components as compositions of very fine-grained sub components associated with specific context conditions and as units of behaviour.

Figure 3 shows a component's architectural template defined in the PrimitiveC XML schema. It shows the component attributes: ID, Name, Location, Mapped Context, Path, Version, and PcomType. Location of the component specifies whether it is stored locally or remotely. The others attribute are straightforward. The second part of the PcomsType represent the components elements.

PrimitiveC-ADL describes PCOMs components in terms of Port, Interface, Type, Semantic, Constraint, Evolution, and Non-functional properties:

- Ports: A port identifies a point of interaction between the component and its environment. A component may provides multiple interfaces by using different types of ports. A port can represent an interface as simple as a single procedure signature, or more complex interfaces, such as a collection of procedure calls that must be invoked in certain specified orders, or an event multicast interface point. A port in PrimitiveC is a complex element, that have multiple interfaces.
- Type: It is a way to encapsulate component's functionality into reusable blocks. Defining a component type could instantiate its implementation into many architecture or it may be used between distributed application.
  A component in PrimitiveC is derived from a context condition with a behavioural action, it may provides or require services.
  The component's instance could participate in one or more application's instances and it could adapt or discover a service or resource. In this context a component type should be defined to increase understandability and analyzability of an architecture.
  The components's attribute PcomeType in Figure 3, is used for this purpose, so we prefer to give the developer a high flexibility in defining component's type.
- Semantics:
  Component semantics define a high level model of a component's behaviour, semantics provide constant mapping of architectures from one level of abstraction to another [4].
  The semantic features in PrimitiveC-ADL are supported by the element DynamicPart as shown in Figure 3. Dynamic Part used to support unanticipated application adaptation based on the captured context.
  A closer look inside the dynamic part of a component is shown in Figure 4. Each dynamic part consist multiple layers and decision points. The structure of the layers have context sensitive elements we call declarative service.
  The element declarative service is used to present the services provided or required by the component. Each service is bound inside the component layer at the development time. At run time the middleware is going to activate or deactivate each layer depending on the running context.
- Constraints: A constraint is a property of assertion about a system or one of its parts, the violation of which will render the system unacceptable to one or more application's instances. Constraints establish dependencies among a component's internal parts [4].
- Evolution: As architectural building blocks, components will continuously evolve. Component evolution can be informally defined as the modication of (a subset of) a components properties, interface, behaviour, or implementation. The modification of a component's subpart (dynamic part) evolves the component's behaviour by modifying the associated implementation by controlling the embedded Variation Point elements as in Figure 3.
- Non-Functional Properties: Safety, security, performance and portability are non-functional properties of a component, these properties can not be derived from the component's behaviour. Early specification of such proprieties enables simulation of runtime behaviour and map component implementation to processors. Non functional properties are mapped in PrimitiveC to the element Static Part.

In Figure 5, a component definition example is illustrated to simplify the understanding of both the component model and thePrimitiveC-ADL, An xml file is directly generated from the PrimitiveC-ADL as shown in Figure 5. A devloper could easily modify the XML file content to describe the application structure. PrimtiveC allows developer to define new architecture elements as needed. This gives high level of flexibility to PrimitiveC-ADL to be used for different domain with multiple requirements.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:Component p:ID="01101" p:Location="Local"
p:Mapped_Context="Lowbandwidth"
 p:Name="ActivateBandWidthComponent"
 p:Path="/workspace/project" p:PcomType="Void"
p:Status="Resolved" p:TimeStamp="2001-12-31T12:00:00"
p:Version="1.0.0"
xmlns:p="http://www.PrimitiveC.org/PCOMs"
xsi:schemaLocation="http://www.PrimitiveC.org/PCOMs
PCOMs.xs">
  <p:StaticPart>
<time>datetime</time>  <user>activeuser</user>
<p:Constraint/>  </p:StaticPart>
  <p:DynamicPart xsi:type="p:DynamicPartType">
    <p:Layers>
      <p:DeclartiveService>
      <p:OS_Calls>getNetworkProvider</p:OS_Calls>
      <p:OS_Calls>GettheBestProvider</p:OS_Calls>
      <p:Resources/>selectwirlessconnection
<p:SensorsPlugIn>executeNetworkseneor</p:SensorsPlugIn>
        <p:Resources/>
 </p:DeclartiveService>
      <p:BundleContext/>
    </p:Layers>
    <p:DecisionPoints>
 <p:VariationPoint>p:VariationPoint</p:VariationPoint>
      <DP>If bandwidth low activate services binder
</DP>
    </p:DecisionPoints>
  </p:DynamicPart>
  <p:Port p:ConnectionType="" p:ID="22"
p:To_Interface="10">
    <p:Interface>p:Interface </p:Interface>
  </p:Port>
  <p:Constraint/>
  <p:VariationPoint>p:VariationPoint</p:VariationPoint>
  <p:Context_Condition>
    <p:Scope>p:Scope</p:Scope>
    <p:Entity>p:Entity</p:Entity>
    <p:Domain>p:Domain</p:Domain>
  </p:Context_Condition>
</p:Component>
```

Fig. 5. A Component Definition Language in PrimitiveC-ADL

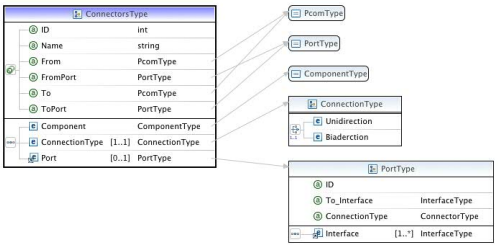

Fig. 6. Connectors Elements

B. Modelling Connectors

Connectors are architectural building blocks used to model and control the interactions between components. In PrimitiveC-ADL, connectors govern the interaction during composition process. A connector is used to connect two or more components in external composition, but a design pattern is used in the internal composition type to replace subdivision of the component by others to satisfy the adaptation.

The connector is characterised by port, types, semantics, constraints, evolution, and connection properties. In this paper we are going describe connectors in the way we use them as artefacts for complex composition mechanisms.

The connector elements are shown in Figure 6. Each connector has a set of attributes, The attributes used by the PCOMS middleware [3], for message passing mechanism, during the adaptation process. The connector's attributes are ID,

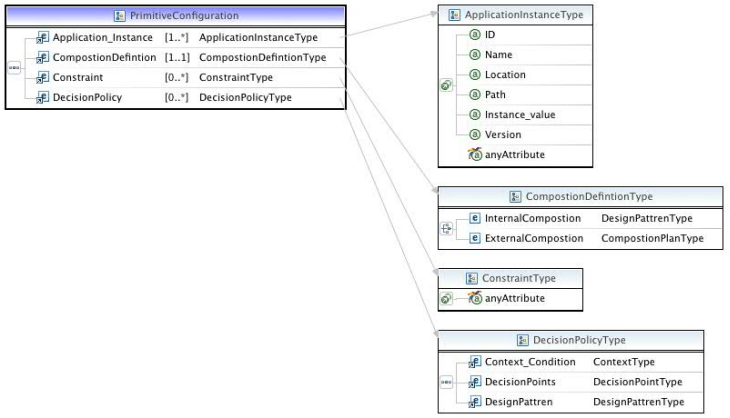

Fig. 7. PrimitiveC Configuration Elements

Name, From, FromPort, To, and ToPort. The attributes From and To used to specify the components, which are involved in the connection.The attaributes FromPort and ToPort used to specify the port id in both components.

The element component represent a set of component which are evolved during the composition process. Connector's port is used to link two or more component as the composition plan specified. The element connection type defines the data flow mechanism. Connection type could be a unidirectional or bidirectional. Finally the constraints are specified to ensure adherence to intended interaction protocols, establish intra-connector dependencies and enforce usage boundaries. The Connector's port is inherited from the template PortType as shown in Figure 6. The port have the element interface which provide multiple interfaces for each port.

C. Architecture Configuration

According to Taylor et. al. [4]. Architectural configuration, are graphs of component and connectors, that describes the software architecture. Configuration description provides more information about the appropriate architecture elemnets.
PrimitiveC-ADL treats Configuration as very important elements. As shown in Figure 7. The configuration's elements are used to describe vital information for the adaptation and composition processes. The Elements of architecture configuration are:

- Application Instance: The Application Instance is used to describe multiple instance of the application. Each composition process produces one application instance. This feature provide a rollback mechanism when the running adaptation process failed. In this case the middleware uses the application instance to retry the adaptation using an older version. A full description about the PCOMs middleware is described in [3].
- Compositionality:

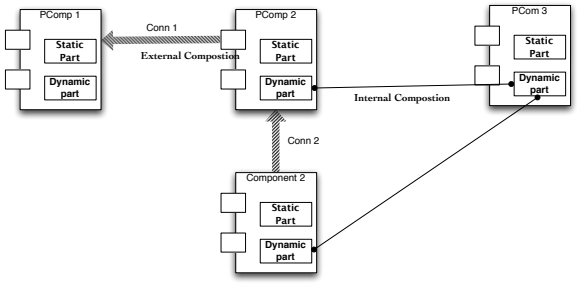

Fig. 8. External and Internal composition

Most ADLs support hierarchical composition of components. PrimitiveC-ADL uses composition elements to clearly describe the composition defintion language (CDL). The CDL is used to define the composition process and the architecture's elements evolvability.
As mentioned above the composition mechanism in the PCOMs component designed to includes two types. The composition definition is an architectural template used to define the external and internal composition. The sub elements of (composition definition Type) are alternatives. Each composition process select just only one element (Internal composition or external composition).
This kind of selection simplified the configuration process, and It is increase the designer understandability. The PrimitiveC-ADL choose which sub elements could be implemented in the new application instance. This feature provide the ADL tools with the active specification, which reduce the space of possible design options based on the current state of the architecture.
The design pattern in the PCOMs is proposed by decision policy and described by the composition plan, that specify the type of composition to be used during adaptation. A composition plan is more specific architecture configuration not a simple method call as in ALI [16].

- Constraints: Constraints specify the dependencies which is complement to PrimitiveC sub elements. Many global constraint could be provided inside the constraint element.

To manipulate the application possible behaviours. Developers design PCOMs components that mapped context conditions. For example (bandwidth, battery and memory). Layers and decision points are embedded in the PCOMs components to support the adaptation. Developers must describe Decision policies in the configuration elements as described above. The following decision policies are predefined and attached to the Decision policy elements.

Decision policy 1: if bandwidth drops to low, change the video resolution and switch from colourful to black and white video.

Scenario 1: The user is doing Skype call with audio and video streaming. Bandwidth drops dramatically to low. The Audio streaming is more important than video for the user to finish the call. In this case the middleware evaluates the

```
<?xml version="1.0" encoding="UTF-8"?>
<p:DecisionPolicy
xmlns:p="http://www.PrimitiveC.org/PCOMs"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.PrimitiveC.org/PCOMs
PCOMs.xsd ">
  <p:Context_Condition>
    <p:Scope>p:Scope="Network"</p:Scope>
    <p:Entity>p:Entity="Bandwidth"</p:Entity>
    <p:Domain>p:Domain="Resources"</p:Domain>
  </p:Context_Condition>
  <p:DecisionPoints>

<p:VariationPoint>p:VariationPoint=VP1</p:Variation
Point>
  </p:DecisionPoints>

<p:DesignPattren>p:DesignPattren=DesignPattren1</p:
DesignPattren>
</p:DecisionPolicy>
```

Fig. 9. Decision Policy 1

predefined decision polices. Decision policy 1 fits the current bandwidth condition (i.e. low bandwidth). The middleware uses the PCOMs component's layer to change the video resolution and the colourful depth to black and white.

Decision policy is described using the following ADL as shown in Figure 9, according to the above decision policy the scenario needs a modification to component 2 implementation. Modifying the component implementation by the PCOMs 3 dynamic part, results configuring the video resolution specified by the decision policy 1. As Decision policy 1 proposed the use of design pattern 1, the design pattern could be described using the elements in Figure 10.

In Figure 8, PCOM1 and PCOM2 are connected using connector Conn1, A unidirectional connection or bi-directional connection depending on the adaptation requirement. Decision policy may be provide further description about the type of connector which could be used, but the best way is to provide a composition plan as element of the architecture configuration, that composition plan describes the whole application architecture with more details about components interaction. The type of ports and the interfaces attached with a specific component could determine the type of connector to be used if and only if this connection is specified by the composition plan. In Figure 8, PCOMs4 is directly connected with PCOMs3 with different type of connector which connects the dynamic part for both components, which modifies their implementation to fulfils the adaptation process requirement.

V. Future Work

In this paper, we propose an Architecture Description Language to support context oriented component. In the future we intend to evaluate PrimitiveC-ADL with case studies. A comparative study should be conducted between PrimtiveC-

```
<?xml version="1.0" encoding="UTF-8"?>
<p:DesignPattren
xmlns:p="http://www.PrimitiveC.org/PCOMs"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.PrimitiveC.org/PCOMs
PCOMs.xsd ">

<p:Component> ID="10" Name="PCOM1" </p:Component>
<p:Context Condition>Lowbandwidth</p:Context_Condit
ion>
<p:DecisionPoints>PCOM1 DP1</p:DecisionPoints>
<p:DecisionPolicy> if bandwidth drops to low,
change the video resolution
 and switch from colourful to black and white
video.</p:DecisionPolicy>
 <p:DynamicPart>dandwidthPCOM</p:DynamicPart>
 <p:Layers>
 <p:ActivateLayer> Voice>
 <VariationPoint = VP1/>
 </p:ActivateLayer>
 <p:DeactivateLayer>Video Resolution/>
 <VariationPoint>black and white</VariationPoint>
 </p:DeactivateLayer>

 </p:Layers>
</p:DesignPattren>
```

Fig. 10. PrimitiveC description of Design Pattern

ADL and other ADLs. A code generation tool to map the generated XML files into an implementation language that suits mobile devices. PrimitiveC-ADL should be tested in several case studies to show its novelty.

An ADL tool that can map the visualisation features provided by PrimitiveC-ADL to an implementation language. The tool must support the XML schema extension produced by this research. PrimitiveC-ADL is just proposed a composition definition language (CDL), more work should test and evaluate CDL in realtime applications. We think CDL is a promising aspect of PrimitiveC-ADL.

Context analysis and reasoning techniques are needed to clearly specify the context aware application requirement. A generic context requirements model is needed to analyse varying context conditions. Anticipating context conditions at an abstract level to map the PCOMs is a challenge.

Policy configuration mechanisms are needed, the mechanisms explain how those policies are designed, processed, and configured to allow new behaviour that was not initiated at design time.

VI. Conclusion

After this comparative study we conclude it is vital to propose new ADL elements to supports context aware applications. Context oriented programming is a promising technique to build mobile applications, due to its capability to accommodate unexpected dynamic adaptation. A composition plan is promising ADL elements beside the use of design Patterns.

PrimitiveC-ADL provides several features by the introduced elements: semantics, evolvability, understandability, flexibility, architectural templates, composition definition language, dynamic adaptation, instantiation, types, and reusability.

PrimitiveC add new architecture's elements to the ADLs domain: layers, composition plan, design pattern, decision policy, declarative services, context sensitive elements (dynamic part), and static part. It provides architecture's elements in two types of template: complex, and simple that increase understandability and flexibility for the developers. It allows developer to add or modify ADL's elements as needed by the application's requirements.

Finally, in this paper we proposed a PrimitiveC-ADL for building context-aware systems. It has many promising aspects, that could support dynamic application adaptation and application's composition.

References

[1] C. Szyperski, D. Gruntz, and S. Murer, "Component software: beyond object-oriented programming," p. 589, Jan 2002. [Online]. Available: http://books.google.com/books?id=U896iwmtiagCprintsec=frontcover

[2] K. Lau and Z. Wang, "Software component models," *IEEE Transactions on Software Engineering*, Jan 2007. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs-all.jsp?arnumber=4302781

[3] B. Magableh and S. Barrett, "Pcoms: A component model for building context-dependent applications," *ComputationWorld 2009*, pp. 44–48, Nov 2009.

[4] R. Taylor, N. Medvidovic, and N. Medvidovic, "A classification and comparison framework for software architecture description languages," *IEEE Trans. Software Engineer*, Jan 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.9588

[5] P. Clements, "A survey of architecture description languages," *IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design*, Jan 1996. [Online]. Available: http://portal.acm.org/citation.cfm?id=858261

[6] P. Oreizy, N. Medvidovic, and R. Taylor, "Runtime software adaptation: framework, approaches, and styles," *portal.acm.org*, Jan 2008. [Online]. Available: http://

[7] M. Mohammad and V. Alagar, "Tadl-an architecture description language for trustworthy component-based systems," *ECSA '08: Proceedings of the 2nd European conference on Software Architecture*, Jan 2008. [Online]. Available: http://www.springerlink.com/index/7587h883r5411472.pdf

[8] E. Dashofy, A. V. der Hoek, and R. Taylor, "A highly-extensible, xml-based architecture description language," *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture*, Jan 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs-all.jsp?arnumber=948416

[9] M. Pinto and L. Fuentes, "Ao-adl: An adl for describing aspect-oriented architectures," *Lecture Notes in Computer Science*, Jan 2007. [Online]. Available: http://www.springerlink.com/index/hq555114x0wt2771.pdf

[10] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The koala component model for consumer electronics software," *Computer*, Jan 2000. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs-all.jsp?arnumber=825699

[11] D. Garlan, R. Monroe, and D. Wile, "Acme: Architectural description of component-based systems," *Foundations of component-based systems*, pp. 47–67, Jan 2000.

[12] C. Atkinson, J. Bayer, and D. Muthig, "Component-based product line development: The kobra approach," *of the First Software Product Lines Conference (SPLC1)*, Jan 2000.

[13] M. Pinto, L. Fuentes, and J. Troya, "An architecture description language for dynamic component and aspect-based development," *GPCE '03: Proceedings of the 2nd international conference on Generative programming and component engineering*, Jan 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=954194

[14] N. Medvidovic, P. Oreizy, J. E. Robbins, and R. N. Taylor, "Using object-oriented typing to support architectural design in the c2 style," *SIGSOFT Softw. Eng. Notes*, vol. 21, no. 6, pp. 24–32, 1996.

[15] R. Anthony, M. Pelc, P. Ward, J. Hawthorne, and K. Pulnah, "A run-time configurable software architecture for self-managing systems," *ICAC '08: Proceedings of the 2008 International Conference on Autonomic Computing*, pp. 207–208, 2008.

[16] R. Bashroush, I. Spence, P. Kilpatrick, T. J. Brown, W. Gilani, and M. Fritzsche, "Ali: An extensible architecture description language for industrial applications," *CBS '08: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pp. 297–304, 2008.