Modeling the Engineering Viewpoint of ODP systems with MODERN

Hai-Quan NGUYEN
Université des Sciences et Technologies de Lille (USTL), 59655-Villeneuve d'Ascq Cedex,France
E-mail: Quan.NguyenHai@lifl.fr.

Abstract

Difficulties in software architecture design come from the lack of analysis tools to assist the architect in detecting system errors. Current approaches in architecture design such as Architectural Description Languages (ADLs) are often limited by the way they tackle quality issues separately and only in one viewpoint. None of them addresses analysis and verification of distributed architecture with a global approach from functional design (i.e. liveliness or safety properties) to its deployment in distributed environments (i.e. the environment provides containers and servers to host functional components), and their non-functional qualities (i.e. performance, reliability).

This paper presents our new language for Modeling the Engineering Viewpoint of ODP systems (MODERN) supporting the integrated analysis for different quality properties of RM-ODP systems. Those qualities range from structural and behavioral properties (such as the assembly correctness and the liveness properties) to the quality of services properties (ex. performance).

1. INTRODUCTION

RM-ODP (Reference Model for Open Distributed Processing) [1,2] is an International Organization for Standardization (ISO) 's standard for modeling distributed systems based on separation of concerns of stakeholders. Although RM-ODP provides rich concepts for modeling distributed systems, it does not provide tools and methods for analysis and verification of software quality properties.

In this reference model, ODP provides five viewpoints where each one represents one concern. The Enterprise viewpoint describes client needs and organization policy, the Information viewpoint describes system data, the Computational viewpoint describes business functionality, the Engineering viewpoint describes mechanism supporting distributed communication and Technology viewpoint describes technology utilization. However, although this reference model provides rich concepts for modeling distributed systems in accordance with the different viewpoints, it does not provide languages and tools for modeling and analyzing software qualities.

This work presents our new language MODERN to specify the technical architecture of ODP system, which is specified in the Engineering viewpoint. Actually, the Engineering viewpoint is one of the most important viewpoints in ODP system. In this viewpoint, concerns such as the distribution, the technical concern should be specified in this level. Quality assurance is important in this phrase, which helps to detect conception faults before the implementation choices.

The MODERN language is expected to provide a number of advantages by supporting the integrated analysis. Firstly, analysis tools can be integrated in the same analysis environment. Secondly, the specifications of different analysis concerns are consistent between them. Third, one unique language is used to address the different analysis concerns.

The paper will be organised as following: in the Section 2, we introduce the motivation of our work. After, we present the MODERN language, which support the integrated analysis capacity in the Section 3. Later, we will illustrate our approach with case study and present quickly our prototype in the Section 4. Afterward, we will compare our work to related work in the architectural modeling. Finally, we conclude and give some perspectives of our work.

2. Motivation & Problems

2.1 Why a new language for Engineering viewpoint?

The engineering viewpoint focuses on mechanisms and functions required to support distributed interactions between components in the system. This viewpoint is important because it addresses the problems of component interactions: how the infrastructure and communication mechanisms support distributed interaction.

In RM-ODP reference model, no languages or tools are defined. The architects are free to choose their own methods. In fact, we can use the languages like UML[9], or the architectural description languages (ADLs)[8] to specify the Engineering viewpoint. However, as we will see in the following, those languages fail to model the different aspects of the ODP's Engineering viewpoint. Moreover, they do not provide sufficient capacity of analysis for the distributed system architecture.

The first choice is highly used in the industry whereas the second choice is studied much in the research field. The UML language, in their latest version 2.0, provides graphic notations for modeling systems. We can access to its dynamic diagrams such as interaction diagram or state chart diagram. However, analysis tools are not supported for UML diagrams. Moreover, concerns such as quality of service assurance are not considered when using this language.

The architectural description languages, in other hand, can provide effective analysis tool. Wright provides analysis tools such as type checking analysis [10], assembly correctness and behavioral verification. However, it does not address the quality of service analysis. Rapide provides simulation capacity but does not provide also this kind of analysis. AADL (The SAE Avionics Architecture Description Language) [31] is a powerful language that provides also a range of the quality analysis such as the dependability properties such as performance or security. However, no means is provided to take into account the new quality property analysis.

2.2 Why do we need integrated analysis ?

We define **integrated analysis** is the capacity of satisfying a well range of quality properties of the system architecture. It is composed of four analysis aspects: the structural aspect analysis, the behavior aspect analysis, the deployment aspect analysis and the quality of service aspect analysis of the component architecture.

The integrated analysis is required when we specify the Engineering viewpoint because it will help us to identify globally the different kinds of potential conception errors of the system. When dealing with distributed system, the architects interest not only in the functionality of the system but also to the mechanism of deployment of the system in the distributed environment.

The integrated analysis can bring many advantages for the architects. The integrated analysis approach has the following qualities:

- Integrated: number of quality properties can be analyzed in the same and unique environment.
- Consistence: specification of the architecture is consistence in accordance to Engineering concepts. Different concepts in the language are verified to assure the consistence between them.
- Unified: One unique language is used to support different analysis aspect of the architecture. The architects reduce their effort in learning different notions when dealing with analysis problems.

2.3 The integrated analysis problems

In our research, we have identified the following analysis problems needed take into account when specifying the Engineering viewpoint:

- Concerning the structural aspect: in this aspect, one of the concerns of the architect is how to assure to interaction between components?
- Concerning the behavioral aspect: we interest in verifying the temporal properties of the architecture. The assurance of those properties validates the functionality of the architecture.
- Concerning the deployment aspect: when deploying the component to the distributed environment, the difficult is to assure the execution of the components in the run-time.
- Concerning the quality of service aspect: the most challenge problem is the integration of new quality of service tool to analyze the technical architecture. The approach is required to be open with new kind of analysis.

MODERN, our modeling language for ODP's Engineering viewpoint, addresses those problems above by introducing not only the concepts supporting the specification of the distributed concerns, but also provide powerful integrated analysis concepts.

3. MODERN: the language for intergrated analysis of the engineering viewpoint

3.1 Overview of the language

The language provides basic concepts for specifying the Engineering viewpoint. Moreover, it provides extra-concepts that support the integrated analysis criterion.

The basic concept of our MODERN language is the component (class *Component Type*). It has the hierarchy structure and is composed of ports (class *Port*), which represent functionality of the component. In the distributed environment (class *Distributed Environment*), the component is deployed to an encapsulation unit called Capsule (class *Capsule*), which has component's life cycle management mechanism thanks to its manager (class *Capsule Manager*).

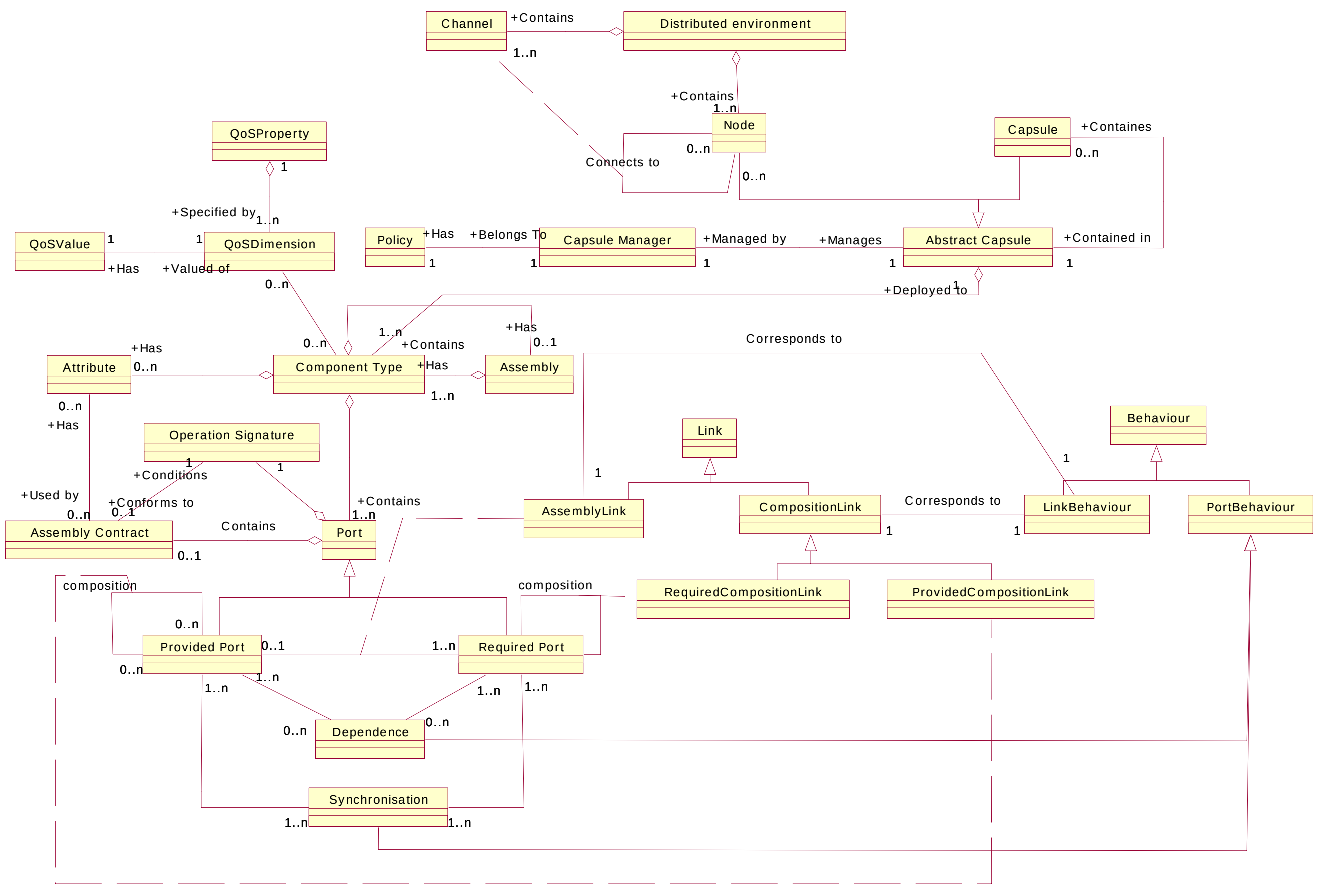

Figure 1: the MODERN meta-model

Besides the basics concepts, the MODERN language provides others concepts in order to archive the integrated analysis. Firstly, we use assembly contract notion helping semantics assembly. Secondly, we provide component's life cycle management to address the deployment analysis. Last but not least, the QoS property is used to integrate new non-functional quality which is the object for the quality analysis.

3.2 Basic concepts

3.2.1 Component

Our architecture is specified with components. The concept of component used in this architecture is based on the following definition from Szyperski[29]: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

The elements needed to describe our component model and the relations between them are taken into account in the MODERN's meta-model represented in Figure 1.

Our approach handles component as component type. Components may have attributes that represent their state. Moreover, they are described by provided and required interfaces. A component communicates with other components through its interfaces. In our model, an interface is represented by a port that is associated with a single service. A service is specified by its signature, which is composed of a name and of ingoing and outgoing parameters.

3.2.2 Components Assembly

The model does not include explicit connectors between components. Nonetheless, if an architect needs one, he can model it in a component. Communications between components or more precisely between their ports are specified by an assembly link. The semantics of a link corresponds to a synchronous call from the required port to the provided port. The choice regarding a port structure involves that a required port can only be bound to a provided port whereas a provided port can be bound to several required ports.

3.2.3 Components Composition

In order to build a complex architecture, we use composite components. They are differentiated from primitive components because they contain subcomponents which may be primitive components and also composite components hence a recursive definition of a component. The ports of a composite component are called delegated ports. Indeed a call to a provided port of a composite component is forwarded to a provided port of one of its subcomponents. Moreover, a call from a required port of a composite component results from the forwarding of a call from a required port from one of its subcomponents.

3.2.4 Component Distribution

In the ODP's Engineering viewpoint, the most important concern is the mechanisms supporting the distribution. Our language, as the result, uses the following concepts to model the deployment of the distributed architecture.

Capsule

In our model, we consider component type entity of deployment. When component types are deployed, we use the capsule concept to encapsulate those components. We manage those components with deployment policy where we specify the management of life cycle's component. We apply the same politic for component in the same capsule.

Node

A node is one of the most important concepts in our meta-model. It is an abstract concept that represents an addressable machine in the reality. The node describes the place where the component types are deployed following the architect choice. The nodes interact between them via communication network. As we consider the node is a special case of the capsule, a capsule manager manages it. Moreover, it can contain other capsules.

Channel

A communication channel represents a communication way between two nodes in deployment environment. Between two nodes, there is only one channel.

This concept is derived from ODP specification. We consider specify the links but not show how to make possible the binding like in ODP's specification. As opposed to channel concept in ODP, ours does not take into account the binding because the communication and interaction problem have been considered in assembly model.

Deployment environment

The deployment environment composes of nodes and channels. It is where the application is deployed and execute. Normally, this environment is distributed.

3.3 MODERN's Integrated Analysis concepts

In our approach, we determine three categories of the integrated analysis. The first category of integrated analysis concept contains the functionality contract concept supporting structural and behavioral analysis. The second category contains component management policy supporting deployment analysis. The last category contains concepts for modeling the quality of service properties and means to integrate them to the components.

3.3.1 Functionality Contracts

The conditions of validity of a component assembly are improved by associating an assembly contract composed of a pre-condition and a post-condition to each port. These conditions focus on the attributes of the component and on the parameters of the signature. Thus in addition to the verification of the signature compatibility between two linked ports, there is an analysis that checks respectively the compatibility of the pre-condition and pos-condition of a port with the precondition and post-condition of the linked port. Furthermore, behavioral contracts are added to the components. These contracts describe the expected behavior of a component and are used to generate the behavior of the components assembly. An appropriate tool has been developed to check some properties on it.

3.3.2 Component Management Policy

Capsule Manager

The components deployed in the distributed environment are controlled by the capsule manager. In fact, the policy which manages the component's life cycle can modify the level of quality of the application. For example, the passivation of the instances of component during it does not have there a request of these customers can make the resources available for other components. Another example is the instantiation of new instance of component or the reactivation of the existing instances for requests these of the customers.

Then we find that it is necessary to specify the management of life cycle of the components deployed in the environment of reception. This specification enables us to check the policy of management of components which influences the level of quality of the application.

Component's Life Cycle Management Policy

For a component type deployed in a capsule, we specify following information for managing the component's life cycle:

- the maximum number of instances of the component type which the capsule manager can instantiate.
- the maximum number of simultaneous requests treated by the component.
- the policy specification of the component's life cycle. This specification must make it possible to specify activation, passivation, and the termination of the cycle of life of the instances of component.

3.3.3 Quality of Service Property

We define the concepts for analysis the quality properties. Three concepts have been proposed: the QoSProperty (QoSProperty), the analysis parameters (QoSDimension) and its values (QoSValue) associated with component descriptions.

QoS Property

The QoS property represent the quality property which one want to analyze.

QoS Dimension

In order to analyze a quality property, we need to have analysis parameters concerned with this property. For example, to analyze the performance property, we need to provide the arrival rate and the service rate of the service center. Other parameters include the number of server and the server capacity.

QoS Value

QoSValue represent the value in a given time of the QoS Dimension. For example, the dimension service rate has the value of 1000 messages/second.

4. Case study

In the following section, we illustrate how to use the MODERN language to model technical concerns in Engineering viewpoint. We will show how integrated analysis is supported by our MODERN language. The language supports **integrated analysis**. Those analyses are composed of the structural aspect analysis, the behavioral aspect analysis, the deployment and the quality of service analysis.

The most significant example verified by our verification is an existing application from the industrial case [30]. The following figure gives an idea of the complexity of the architecture. Required and provided ports are symbolized respectively by - and +. The example represents the use of a generic coupler of scientific code[30]. This coupler is used to study the interactions between codes of different domains in physics. It manages the exchange of values between the codes. The assembly model is described graphically in the figure 4. Two scientific code components Tree and Support communicate between them via the coupler. We need also to analyze globally the qualities properties of the system.

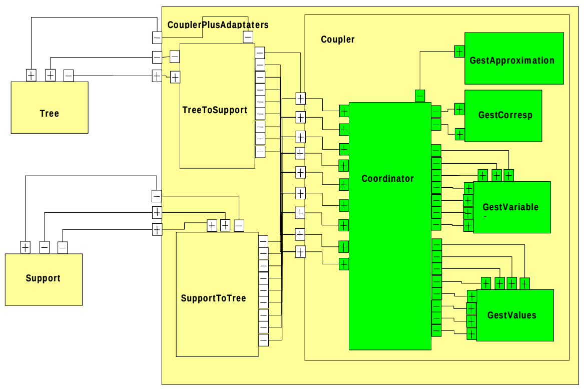

Figure 2: the Coupler application – the assembly view

The following specification using MODERN describes partly the component Tree. It has three ports: two provided port and one required port. In the specification of the Position port, we describe the assembly contract with the pre and post description. In the pre condition of the contract, we specify that the position of x must be superior of 0 and inferior of 100 and the position of y must be in the range of 0 and 100 also.

```
<Component_Type Component_Name="Tree">
   <Attribute attribute_name="att_position_x" type="float"/>
   <Attribute attribute_name="att_position_y" type="float"/>


…

 <Provided_Port port_name="Position">
              <Signature name="void Position(OUT float position_arbre_x,
OUT float position_arbre_y)"/>
              <Assembly_Contract>
      <Pre
expression="position_x<=100&&position_x>0&&position_y>0&&position_y<=100"/
>
                <Post
expression="position_arbre_x==position_x&&position_arbre_y==position_y"/>
              </Assembly_Contract>
 </Provided_Port>
……
</Component_Type>
```

Figure 3: the Position provided port specification

The links between the components are specified in the figure 4. We show that the Tree component requires the Force port of the Coupler component for its execution whereas the Coupler requires the Speed and the Position port.

```
<Assembly_Link>
        <Required_Port Component_Name="Tree" Port_Name="Force"/>
        <Provided_Port Component_Name="Coupler" Port_Name="Force"/>
   </Assembly_Link>
<Assembly_Link>
        <Required_Port Component_Name="Coupler" Port_Name="Speed"/>
            <Provided_Port Component_Name="Tree" Port_Name="Speed"/>
</Assembly_Link>
<Assembly_Link>
  <Required_Port Component_Name="Coupler" Port_Name="Position"/>
   <Provided_Port Component_Name="Tree" Port_Name="Position"/>
</Assembly_Link>
```

Figure 4: the assembly links between the components

The figure 5 shows the graphic representation of the application in the distributed environment. The Tree component is deployed into the node NodeA, the Support component into the node NodeB and the Coupler into the node NodeC.

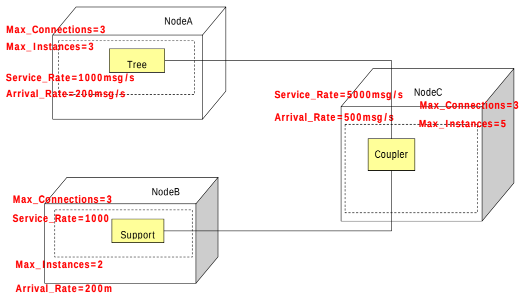

Figure 5: the Coupler application - the quality of service analysis view

As we interest in the performance quality analysis, the following section describes the analysis information for the performance quality. Using MODERN, we describe this quality by four parameters:

- The customer service time distribution (*Service*)
- The customer inter-arrival time distribution (*Arrival_Rate*)
- The number of independent component instances (*Max_Instances*)
- The queue capacity (*Max_Connections*)

The figure 6 describes the technical architecture containing three nodes A, B and C. In the node A, the component Tree is specified with the following performance parameters: the service rate equals to 500msg/second; the arrival rate equals to 100msg/second). The component Support is deployed in the node B with the following performance parameters: the service rate equals to 1000msg/second; the arrival rate equals to 200msg/seconde). The component Coupler is deployed in the C with the service rate equals to 5000msg/second and the arrival rate equals to 500msg/seconde).

```
<Node NodeName= "NodeA">
  <Capsule CapsuleName= "CapsuleTree">
     <Contains_Component Component_Name="Tree">
            <QoSProperty QoSProperty_Name="Performance">
           <QoSDimension Dimension_Name="Service_Rate" Value="500"/>
           <QoSDimension Dimension_Name="Arrival_Rate" Value="100"/>
        </QoSProperty>
     <Contains_Component/>
     <CapsuleManager Manager_Name="Tree_Manager">
            <Policy>
                 <MaxInstances Component_Name="Tree" Value="3"/>
                 <MaxConnections Component_Name="Tree" Value="5"/>
            </Policy>
     </CapsuleManager>
  </Capsule>
```

```
</Node>
<Node NodeName= "NodeB">
  <Capsule CapsuleName= "CapsuleSupport">
     <Contains_Component Component_Name="Support">
            <QoSProperty QoSProperty_Name="Performance">
           <QoSDimension Dimension_Name="Service_Rate" Value="1000"/>
           <QoSDimension Dimension_Name="Arrival_Rate" Value="200"/>
        </QoSProperty>
     <Contains_Component/>
     <CapsuleManager Manager_Name="Support_Manager">
            <Policy>
                 <MaxInstances Component_Name="Support" Value="2"/>
                 <MaxConnections Component_Name="Support" Value="2"/>
            </Policy>
     </CapsuleManager>
   </Capsule>
</Node>
<Node NodeName= "NodeC">
  <Capsule CapsuleName= "CapsuleCoupler">
     <Contains_Component Component_Name="Coupler">
            <QoSProperty QoSProperty_Name="Performance">
           <QoSDimension Dimension_Name="Service_Rate" Value="5000"/>
           <QoSDimension Dimension_Name="Arrival_Rate" Value="500"/>
        </QoSProperty>
     <Contains_Component/>
     <CapsuleManager Manager_Name="Coupler_Manager">
            <Policy>
                 <MaxInstances Component_Name="Coupler" Value="3"/>
                 <MaxConnections Component_Name="Coupler" Value="7"/>
            </Policy>
     </CapsuleManager>
   </Capsule>
</Node>
<Channel Channel_Name="ChannelAB"/>
<Channel Channel_Name="ChannelBC"/>
<Channel Channel_Name="ChannelAC"/>
<Distributed_Environement>
            <Connects_To  Node_Name=”NodeA”  Node_Name=”NodeB”  Channel=
“ChannelAB”/>
  <Connects_To Node_Name=”NodeB” Node_Name=”NodeC” Channel= “ChannelBC”/>
  <Connects_To Node_Name=”NodeC” Node_Name=”NodeA” Channel= “ChannelCA”/>

</Distributed_Environement>
```

Figure 6: Engineering specification

Behavioral analysis

In order to analyze on the behavior of a component model, we transform the behavioral elements into FSP (Finite State Process)[25] processes. We operate this translation first by generating a behavior formed of the behaviors of the composition and assembly links and the dependences and then by making a composition of it with the synchronizations.

The analysis uses a verification tool for concurrent systems, named LTSA (Labelled Transition System Analyser) [26], which supports FSP and a LTL (Linear Temporal Logic) checker to check safety and liveness properties such as deadlocks or absence of reachability.

Deployment analysis

The deployment analysis contains two tests: the validity of the deployment and the component's life cycle management analysis. The first analysis ensures that the components, which are assembled on the assembly, can communicate between them after the deployment process using mechanism provided by the environment. The second analysis verifies the execution correctness of the component in the run-time.

We define this validity a correspondence between technical architecture to be deployed and the distributed environment. As we explain above, we must ensure the correspondence between components to be deployed and the distributed environment.

Quality of service Analysis

In order to analyze the performance, we transfer the description of performance quality to queuing network model. The following section describes the results obtained from the analysis phase. In fact, the result is obtained by using the MCQueue tool.

5. Related work

In the following section, we discuss approaches in modeling the architecture of systems and its analysis methods.

5.1 ODP's related work

There are some efforts to define the language of specification from the Enterprise viewpoint with the UML [3]. In this proposal, it describes the concepts in the viewpoint of Enterprise like the policy, the roles, the community. In other work [4], OCL is used to describe the obligations of objects of company. Romero et al.[5] proposes the use of Maude, a logical language of achievable rewriting (executable rewriting logic), to specify the Computation viewpoint. The rewriting logic is the logic of change which can work with the state and calculations non-determinist. Maude is used to specify the concepts from the Computational viewpoint like the object, the interface, the behavior, the constrained ones of environment.[6] This work aims at formalizing the Computational viewpoint of by proposing the concept action template and the causality control.

In spite of the interesting proposals, we find that these approaches are limited to the languages for the Enterprise viewpoint of and the Computational viewpoint. Other viewpoint such as that Engineering is not treated for this moment.

5.2 Architecture Description Languages

Specifying software architecture requires an architectural language to define all static and dynamic aspects of architecture. These languages define the formalism in terms of component, connector, and configuration.

This clear separation between components, connectors and configuration is a basic concept shared by all ADLs. However, despite a large popularity among the research community, no ADLs have yet reached common practice in companies. However, several ADLs have been introduced like ACME, ArchJava, Aesop, C2, Darwin, Meta-H, Olan, Rapide, SADL, UniCon, Weaves, Wright, or xADL. Medvidovic and Taylor [7] provide a general framework to classify and compare them.

One of the advantages of ADLs is the way the associate with formal techniques. Traditionally, those methods are considered difficult to apply in company environment. They do not provide enough notions and tools to associate with the architecture notion. Besides, there is a lack of methodology and software tools. To overcome this drawback, some ADLs language use formal method as tool for analyzing properties. Wright use CSP for analyzing behavioral properties such as deadlock. Darwin use pi-calculus and later use FSP to analyze the liveness and safety property.

5.3 Quality of service analysis

In [11], Issarny and al, interest in dependability analysis from architectural viewpoint. Their work is based on work in ABAS[12] for dependability system. This work is closed to our problem. However, as there is no separation between functional and distribution concern, it is more difficult to determine where the faults come from. The functional architecture uses the connector which differentiates from our approach which considers it is only a specific component. Moreover, the approach does not provide iterating steps to construct technical architecture as ours.

Other quality analysis method such as queuing network for performance analysis [13,14,15,16,17] or block diagram[18,19] for reliability analysis provide efficient means to analyze qualities. Those approaches need to be integrated into architectural level to facilitate the design and analysis of those non-functional qualities.Some efforts have been made in the field such as the integration of performance technique with the ADL [20,21]. However, those approaches treat only one aspect and only one non-functional quality.

6. Conclusion

In this paper, we present our new language MODERN to specify the Engineering viewpoint of RM-ODP systems. The language provides a high effective manner to satisfy the quality properties of distributed software architecture.

The language has a number of original points:

- It provides rich concepts for specifying Engineering viewpoints of distributed system. Those concepts, which conform to RM-ODP standard, help to model the different concerns such as the functionality and the deployment of the system.
- It supports the global approach for analyzing quality properties. Those qualities contain the functionality properties (structural and behavioral properties of components assembly), the deployment correctness and the quality of service property.

- It is provided with a modeling and analyses environment. Number of properties can be validated thanks to our integrated environment.

In our further work, we are studying a new methodology for automatic transformation from the Computational viewpoint. We are working on the new transformation language that uses the architectural figure concept [22, 23, 24] to generate the Engineering viewpoint from a computational viewpoint specification.

ACKNOWLEDGEMENTS

This research has been originally supported by Electricité de France. The author thank to Professor Laurence Duchien of LIFL, Mr Philippe Bedu, Mr Jean Perrin of EDF, Professor. Duong Vu of Eurocontrol and Professor. Thuc Nguyen Dinh of HCM university for the previous discussions and the comments.

REFERENCES

[1] ISO/IEC. Open Distributed Processing Reference Model - parts 1,2,3,4, 1995. ISO 10746-1,2,3,4 or ITU-T X.901,2,3,4.
[2] J-R. Putman, Architecting with RM-ODP, Prentice-Hal, 2001.
[3] X. Blanc, M.-P. Gervais, R. Le-Delliou, Using the UML Language to Express the ODP Enterprise Concepts, LIP6 1999/024: Technical Research, LIP6, 1999.
[4] J.Putman, Model for Fault Tolerance and Policy from RM-ODP Expressed in UML/OCL, Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2000.
[5] R.Romero, A.Vallecillo, Formalizing ODP Behavioural Viewpoint Specification in Maude, EDOC, 2004.
[6] R.Romero, A.Vallecillo, Action Templates and Causalities in the ODP Behavioural Viewpoint, Workshop on ODP for Enterprise Computing (WODPEC 2004), 2004.
[7] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.
[8] N.Medvidovic, R.N.Taylor, A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transactions on Software Engineering, January 2000.
[9] [OMG/UML] OMG, OMG Unified Modeling Language Specification, version 2.0, http://www.omg.org/cgi-bin/doc?ptc/2004-10-02 , 2004.
[10] R. Allen, A Formal Approach to Software Architecture, PhD thesis, School of Computer Science Carnegie Mellon University,Pittsburgh, Mai 1997.
[11] V.Issarny, C.Kloukinas, and A. Zarras, Systematic aid for developing middleware Architectures, Communication of ACM, Vol 45, Num 6, June 2002.
[12] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, H. Lipson, Attribute-based architecture styles, Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), 1999, 225-243.
[13] K.S. Trivedi, Probability and statistics with reliability, queueuing, and computer science applications, John Wiley & Sons, 2001.
[14] H.G. Perros, Queuing networks with blocking, Oxford University Press, 1994.
[15] K. Kant, Introduction to computer system performance evaluation, McGrawHill, 1992.
[16] L. Kleinrock, Queuing systems, Wiley, 1975.
[17] S.S. Lavenberg, Computer performance modelling handbook, Academic Press, 1983.
[18] G. Myers, Software reliability - principles and practices, John Wiley and Sons, 1976.
[19] NASA, Reliability block diagrams and reliability modelling, Technical report, NASA Glenn Research Center, May 1995.
[20] H.Garavel,H.Hermanns, On combining functional verificaition and performance evaluation using CADP, INRIA research rapport, 2002.
[21] S.Balsamo,M.Bernardo,M. Simeoni, Performance evaluation at the software architecture level, SFM (School on Formal Methods), LNCS 2804, 2003.
[22] H.Q Nguyen, P.Bedu, L.Duchien, J.Perrin, H.M.Tran, Architectural Figure – a Reuse Pattern for Analysis and Transformation of ODP Based-Systems, IEEE International Conference on Information Reuse and Integration (IEEE IRI-2005), Las Vegas, Nevada, USA.
[23] H.M.Tran, H.Q Nguyen, P.Bedu, L.Duchien, J.Perrin, Figures de Transformation pour des Architectures Logicielles, LMO, 2005.
[24] H.Q Nguyen, L.Duchien, P.Bedu , J.Perrin, Achieving technical architecture with architectural figures, Proceedings of the IASTED International Conference on Software Engineering and Applications, Cambridge, USA, 2002.
[25] J. Maggee and J. Kramer. Concurrency - State Models and Java Program. John Wiley & Sons, 1999.
[26] J. Maggee, J. Kramer, and D. Giannakopoulou. Behaviour analysis of software architectures. In Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSA1), 1999.
[27] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. Lopez, and G. Puebla. The ciao prolog system: A next generation logic programming environment. Technical Report 3/97.1, CLIP, April 2004.
[28] G. Leavens and Y. Cheon. Design by contract with jml. Draft paper, March 2004.
[29] C. Szyperski, Component Software - Beyond Object-Oriented Programming, Addison-Wesley, 2002, ISBN 0-201-74572-0.
[30] C. Caremoli and J.-Y. Berthou. CALCIUM V2: Guide d'utilisation.
[31] Feiler, P.; Lewis, B.; & Vestal, S. "The SAE Avionics Architecture Description Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering. " RTAS 2003 Workshop on Model-Driven Embedded Systems, May 2003.
[32] Java API for XML Processing (JAXP), http://java.sun.com/xml/jaxp
[33] MCQueue tool, http://staff.feweb.vu.nl/tijms/