Developing a Common Operating Environment for Military Application *

Jungyoon Kim, Joon-Sang Lee, Doo Hwan Bae
Division of Computer Science,
Korea Advanced Institute of
Science and Technology,
Taejon 305-701, Korea
{*jkim, joon, bae*}*@se.kaist.ac.kr*

Dong-Kuk Ryu, Sang-Il Lee
Agency for Defense Development,
P.O.Box 132,SongPa-gu,
Seoul 138-600, Korea
dkryu@lycos.co.kr, happyjoy@lycos.co.kr

Abstract

Interoperability and reusability are major issues in large-scale software system development. Military applications, one of such large scale software systems, have utilized the Common Operating Environment (COE) for the last decade. The COE reduces duplicated development effort and enhances the reusability/interoperability by providing standardized infrastructure and development guidelines. The COE is an intrinsically distributed computing environment, and needs a various techniques in realization. Generally, widely accepted techniques in industry and academic are practical for such environment to implement. We performed a development of the COE pilot project, and in this paper we cover some issues on the COE realization and show certain ways for practical implementation.

1 Introduction

Interoperability and reusability become major issues that software developers need to consider during development, especially in large-scale software system development. The military domain of large application has received much attention, and the common operating environment (COE) has been established for the domain by each country and utilized for the last decade. In case of USA, the Department of Defense suggests software developers to follow a guide, Integration and Runtime Specification (I&RTS[5, 6]), and many military applications (such as C4ISR, GCCS, etc.) have been developed on the top of the COE.

The COE is considered as a key concept for cost reduction of application developments and effective application operations. It reduces overlapped development effort by providing standardized infrastructure that is commonly used by various applications, and enhances the reusability/interoperability by recommending unified architecture and development guidelines.

The issues on COE can be summarized in five categories (explained in section 3) as: construction of COE itself, application development technique, inter-operability at runtime, concrete development process, and compliance level check. The COE issues are also related to the distributed system and DSSA (Domain Specific Software Architecture [2, 3, 4]). The COE provides standard interfaces to various applications on heterogeneous and distributed computing environment. It is related to distributed and real-time systems and is technically addressed by many researchers[1], but we focus on somewhat different viewpoint related to large enterprise information systems. From the reusability viewpoint, the COE concept is very much the same as the one of the DSSA in that both confine a problem area into a specific domain that has to be solved. A military application on the COE is usually characterized as a kind of DSSA, that is not algorithmically complex, but very large scale distributed software with repeated processing units.[1]

We have tried to create a simple paragon of the COE. One of our goals is to realize an efficient way of building the COE using prevailing techniques today. After examining the I&RTS we decided to adopt Java based techniques that are advantageous in building large scale enterprise applications. Other techniques could be chosen among number of nomminees such as super vendor's COM/OLE or OMG's CORBA, but we preferred Java because of its strong support for enterprise features.

We defined a practical architecture model and a development process for building applications on top of the architecture. The architecture is a layered style based on the Java technique. The important element of the COE is the *segment* that can be a software or data unit much like (but not

*This research project has been partially supported by ADD(Agent for Defense Development) and SPIC(Software Process Improvement Center).

[1] The Command & Control (C2) and the successor, C4ISR, are the examples.

Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)
0-7695-1910-5/03 $17.00 © 2003 IEEE

the same as) the component in the concept of Component Based Software Engineering (CBSE). To realize the development process, Rational Unified Process (RUP)[11] has been employed. RUP treats collaborations [2] among objects as the basic way to organize development process(work flow). We also considered collaboration based approaches [7, 8, 10] to realize the segment compositions of the COE.

This paper shows our experiences in realizing the COE prototype following I&RTS guidelines. The sections 2 and 3 explain about the COE and issues on it. The section 4.4 summarizes our efforts to implement the prototype and some problems observed, the section 5 summarizes whole point of our work and future work.

2 Overview of DII COE

The main purposes of the COE include, 1) cost reduction of development through software reuse eliminating stovepipe systems in which almost same infrastructure is repeatedly developed for each of the systems, 2) promote interoperability among applications and increase usability of software.

The DoD of United States forces software developers to use a framework for the creation of a set of cooperating computing environment, the "Defense Information Infrastructure (DII) Common Operating Environment (COE)." The DII COE [3] is a *layered architecture* having four layers:

- Layer 4: Mission applications.
- Layer 3: Common support applications
- Layer 2: Infrastructure services
- Layer 1: DII COE Kernel

The lower layers from the layer 1 to 3 are called the COE and it is the infrastructure that reduces redundant and repeated development/existence of multiple business applications. In the DII COE, the lower levels(COE) are shared by all applications and I&RTS strongly recommends to include commercial off the shelf software (COTS). DISA puts the list of available COTS on its website. The topmost layer (layer 4) is the actual business applications, so called Mission Applications, and any of them does not need to build its own infrastructure. The Fig. 1 shows the COE architecture describe in I&RTS.

The fundamental element of DII COE is the *segment* that is a collection of software or data, and a basic building block in development of systems in DII COE. A segment can be implemented as a component or as a library

[2]Collaboration: a set of interactions among classes for a given set of purposes

[3]The abbreviation DII COE is practically identical to the COE

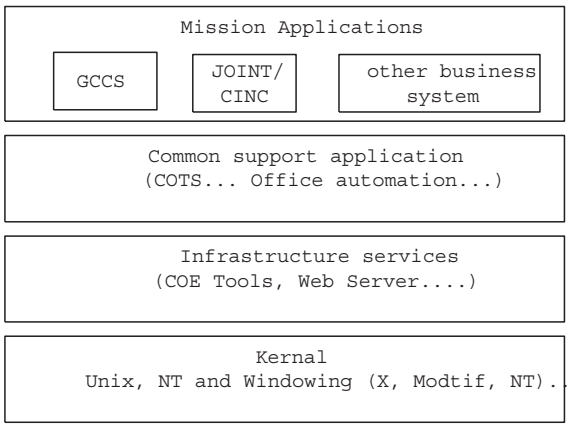

Figure 1. The architecture from I&RTS

that may provide operating system interfaces. Any software building technique can be used to implement the segment, however, it must follow the requirements of I&RTS to be a segment. Segment must provide any self-descriptive functionality that will be used for checking compatibility with the COE and for being retrieved when a developer tries to find reusable segments.

A segment is different from a component. First, a component can be characterized as an independent deployment and implemented separately from interfaces clearly, while a segment is a collection of any software and data units. Second, a component describes itself through interface contracts, while segment describes its various kind of information through a descriptor. Finally, a component is mainly based on object oriented technique such as JavaBean, EJB, COM, etc, while a segment can be any type of collection of software or data as log as it keeps the guidelines of I&RTS. So a component can be a segment if it is accompanied by a descriptor sufficing the requirements of I&RTS.

In 1994, DISA(Defense Information System Agency) of DoD began development of the "Global Command and Control System (GCCS)" and the GCCS became the first system utilizing the COE. After that, more than 100 systems are successfully developed on top of the DII COE. The DoD is still underway to deliver the newer version of the DII COE with the guideline document, "Integration and Runtime Specification (I&RTS)." In addition, Great Britain has launched a number of defense system developments , such as C2I(Command, Control and Intelligence) and FATP (Finance, Administration, Training and Personnel), under the guideline of DIA (Defense Integration Architecture), the UK version of the COE.

3 COE Based Development Issues

The important issues related to the COE can be summarized in five categories, 1) construction of the COE itself, 2) segment development, 3) inter-operability at runtime, 4) concrete development process, and 5) compliance level check.

Construction of the COE itself: To achieve the goals of the COE, a developer should give his/her careful consideration. First of all, he/she must define a segment as an important element of the COE and the architecture of the COE considering interoperability and reusability by following the guideline of I&RTS. He/she also needs to define a segment development process that is the concrete instance of the process in I&RTS. For the definition, he/she may prefer to select de-facto standards techniques in industry and to avoid immature leading edge technology that may occur undesirable effects on COE[12]. Middleware is important in distributed system development, and selecting one among multiple nominees is another issue. Developers take reusable services provided by standard or proprietary middleware infrastructure, such as CORBA, DCOM, Java RMI, etc. He/she will select a proper middleware to compose the COE to deal with nonfunctional requirements for distribution, security, transactional processing, and fault tolerance. This issue is address in section 4.1.

Segment development method and technique: Since the utility of the COE is to place mission applications on it, developing interoperable and reusable segments of mission applications is also important. For improved reusability of segments, they must be totally handled with well-defined interfaces as black-box components, and also support separation of concerns between intra-segment and inter-segment behavioral parts. Employing architectural consideration to the COE mission applications can largely help the above issues. We plan to extend the COE by using the enhanced role model [19] as an architectural style. To acquire appropriate segments, each segment must provide the information of functionality as well as that of interfaces, with which the architect can retrieve in the segments library of a COE. We are developing a mechanism to guarantee the behavioral soundness of segments based on design contracts [16] and subtyping [13]. Well-defined interfaces bring in not only a good reusability and also interoperability. Currently, the COE allows for only the Java delegation-based event model, so the style of segments composition and interoperation is fixed to publisher-subscriber [17]. The support for higher-level architectural styles such as pipe-and-filter and Chiron-2 (C2) [18] is expected to yield better reusability and interoperability. Part of this issue is addressed in section 4.2.

Inter-operability at runtime: After being installed, a segment can be used by other parties using the COE. When we build up a mission application out of preexisting installed segments, it is just a simple case of reuse. Two isolated mission applications are only allowed to communicate with each other via DBMS. This degree of interoperability is very primitive in aspects of performance and safety. For instance, it is very typical that a newly developed mission application for missile defense is expected to cooperate with a running mission application for multi-purpose radar detection. To do this, all running and public mission applications in a COE must be allowed to be monitored and composed for tight cooperation at the segment level, dynamically. This degree of interoperability can support more abstract and extensible composition mechanism than TCP/IP-based cooperation. We plan to develop a manager for dynamic inter-operability between mission applications at the infrastructure services layer of the COE. The runtime interoperability manager consists of three segments: running-segment monitor, running-segment composer, and GUI-based tool for integrating mission applications. We do not address this issue but we are working on it for future work.

Concrete development process: I&RTS defines segment development process guiding developers to keep building valid, interoperable and reusable segments. Because the process avoids dictating a particular process, developers must define a concrete process keeping the minimum requirement of I&RTS. I&RTS requires to make the process to have a concept of "automated integration" encouraging use of automated tools. It also requires supervision of DISA who assesses developers' activities to make them follow the guidelines of I&RTS. Moreover, the developers need to realize the development process considering the system characteristic or confronted situations. This issue is addressed in section 4.3.

Compliance level check: I&RTS suggests the concept of compliance level check for a mission application system including COE. The compliance level check is to evaluate the extent that whether the system is well developed and the COE for the system is well defined and constructed faithfully following the guideline of I&RTS. Even the compliance checklist in I&RTS can be used in reality, developers still need to make it more concrete according to each case. Of course, it should basically fit for the purpose of I&RTS. That is to say, the checklist in I&RTS includes items that properly point out the requirement of low level elements such as communication protocols (i.e., TCP/IP) or some standards (i.e., POSIX), but it does not have explicit items for high level requirements. The high level items of the checklist should address how well segments are reusable and interoperable guaranteeing the safety or completeness of segments composition. We performed compliance level check applying the checklist of I&RTS as it stands, but, this issue is actually closely related to the previous issues to be properly addressed including extension of the current

checklist, and we leave this for future work.

We tried to cover above issues in our work. Our effort includes building a pilot project under the philosophy of DII COE based on Java techniques. This experience is the result from a six months project entrusted by a research institution [4] to examine the issues in realization of DII COE and to find a better way to implement the DII COE mission applications. For that purpose, we followed the steps, 1) define a segment as a basic building block following the guidelines of I&RTS, and the COE architecture based on Java techniques, 2) define the segment development process under the guidelines of I&RTS and MIL498, with support from SPIC[5]. 3) develop two mission applications consecutively in the way that developers utilize the simple tools provided by COE, and they try to reuse the segments from previous application to develop later one. Through the project we properly covered the issues on COE construction and process definition, but we are still hanging on the works of segment development techniques and compliance level checking.

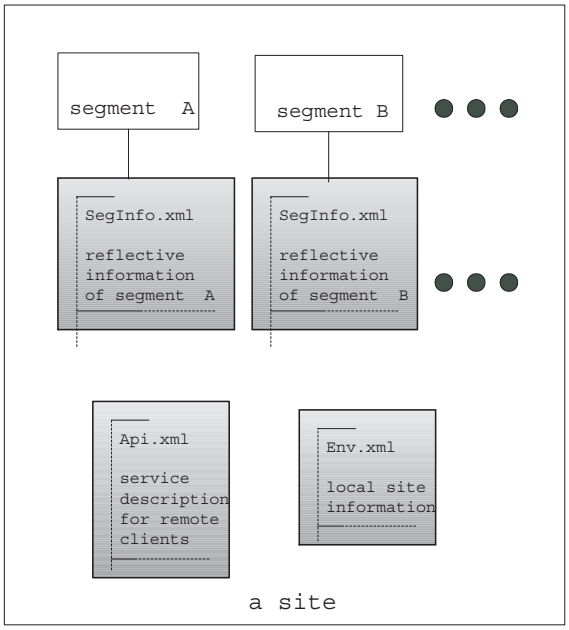

Figure 2. Segments and descriptors on a site

4 COE implementation: Our experiences

4.1 Segment Definition

To define a segment, we exploit the Java technique. Basically, the segment in our project is a Javabean or an EJB. I&RTS defines that a segment should follow guidelines, such as following naming convention and having descriptor for the segment's self-description. So, each of our Javabeans and EJBs has its own descriptor file. I&RTS gives an example of descriptor that is a text file having tags and attributes, but XML file is better because of its easy manipulation. That is to say, we customized the segment concept based on JavaBeans and EJBs in the guidance of DII COE goals. The Fig. 2 shows an example of simple layout of segments and descriptors on a local site.

In the Fig. 2, there are three kinds of descriptors. Each has particular information for its own purpose as next.

- `Env.xml`: local site information such as installed segments list, names of segments and executable files for segments.
- `Api.xml`: server information such as provided services to remote clients. This descriptor is placed on server mode site.
- `SegInfo.xml`: accompanies a segment describing the segment such as its required/provided services or names of executable files, etc.

[4] Agency for Defense Development (ADD), Korea

[5] Software Process Improvement Center, a research organization supported by Ministry of Information and Communication, Korea

Next shows the DTD of `Env.xml`. The other two are omitted for simplicity. The xml files are used by the COE automated tools for checking conflicts among segments and retrieving reusable segments or services provided by segments.

```
< !DOCTYPE env [
< !ELEMENT env (installedSegs)>
 <!ELEMENT installedSegs (Seg)*>
  <!ELEMENT Seg (Direct, ExecutableFileList)>
   <!ELEMENT Direct (#PCDATA)>
    <!ELEMENT ExecutableFileList (file)*>
     <!ELEMENT file (#PCDATA)>
     <!ATTLIST file derictory CDATA #REQUIRED>
    <!ATTLIST Seg name CDATA #REQUIRED>
]>
```

We also defined the COE architecture based on the Java techniques, mainly the Java 2 platform Enterprise Edition (J2EE). The document "DII COE Java Development Guidelines [6]" from DISA guides developers by surveying Java techniques and recommending developers' technical practices that can be helpful for developers to meet COE philosophy. The Fig. 3 shows the architecture we defined under the guideline of I&RTS.

4.2 Segment Composition

In the COE, a segment should be installed and removed automatically as Plug and Play manner. The COE provides

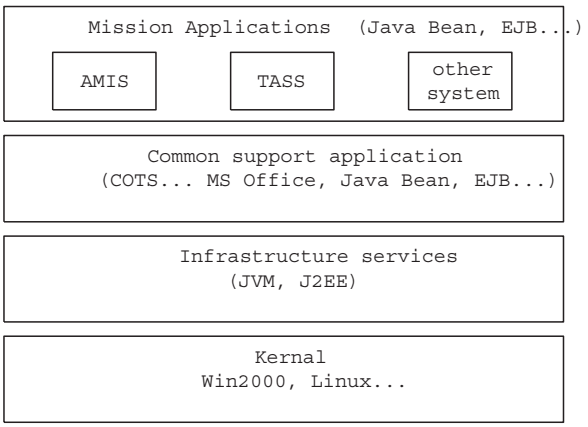

Figure 3. The architecture based on Java techniques

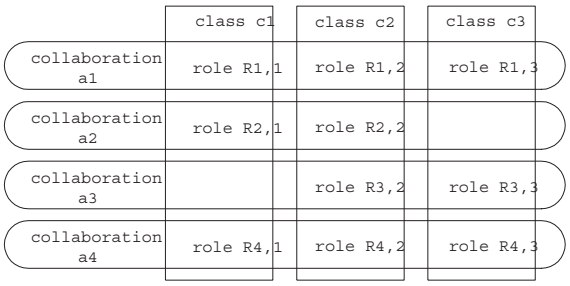

Figure 4. Collaboration-based decomposition (rewritten the Mezini's work[9])

a number of automated tools that can be helpful for such automation. I&RTS suggests more than 40 tools. The major tools among them such as `VerifySeg`, `COEInstaller` and `COEScanCOTS` are working on descriptors to validate environment, segment install and compositions.

For sound composition of segments, developers should pay attention to behavioral properties. Many techniques (i.e. Adaptive Plug-and-Play Components [9], contract based, subject oriented programming, mixin layers[7], etc.) have proposed for that reason and those are referred to as collaboration based development[9] techniques. The segments exist as various types of software units such as EJB, DCOM or some collection of executable or data files, on distributed environment such as Internet or direct cable on the front line in warfare. This heterogeneity brings about problems of segments composition, which can be solved by collaboration based techniques. Descriptors provide the segments' behavioral information. The descriptors may be read through a webserver by automation tools, then will be used to make collaborations (segment compositions).

Since mission applications on the COE are not algorithmically complex, but very large scale distributed software with repeated processing units, we choose the approach of Mezini [9]. Among the various techniques, role concept fits the characteristic of mission applications on the COE. Of course, other techniques can be used according to any peculiarity of application character.

The Fig. 4 describes the basic concept of role and collaboration. Each class (small rectangles) can be regarded as a segment and each collaboration can be a segment composed of several sub-segments. The vertical relationship between two classes may be the inheritance. Suppose we originally composed class `c1, c2, c3` creating the collaboration `a1` in the figure to implement a service. A class that takes a role in a collaboration usually takes a different roles in other collaborations. Such idea can be realized with mixin layer [7] using C++ template and Smaragdakis[8] shows a clear example. Unfortunately, our project is based on Java, implementing mixing layer is quite difficult because Java does not have strong support for multiple inheritance like C++, so we are still hanging on the work.

An example of composition mechanism has been introduced above. Now we need to consider a way to guarantee the safety of such composition. Liscov[13] classifies such problem in his work as *safety* property and *liveness* property. The safety property can be explained as "nothing will not happen," while the liveness property as "something good eventually happens." For the safety property, number of researchers including Findler[14] address it very well. On the other hand, for the liveness property, a mature and simple solution has not been addressed even there are some related works such as Schrefl[15].

We have tried to include such behavioral information into the descriptor of a segment, and that is basically based on the contracts[16]. However, the status of our work is still on going. We still try to make a clear example of contract implementation, and also search an effective way to include the preliminary concept of liveness property.

4.3 Process

The I&RTS suggests a process for segment development. In that process, DISA controls project managers who are in charge of developments of mission applications that are composed of segments. The purpose of the control is to encourage the developers to design reusable and interoperable software. The realization of our development process is based on the "MIL498," the process standard of DoD of USA, and the "Simplified RUP," a development method that nicely meets the requirements of MIL498.

The segment development process in I&RTS has five steps. Each step can be explained as follows:

- Registration : a project manager submits the project development plan and acquire approval from DISA. At this step, developers download the COE and reusable segments previously built. For launching a process, the COE must be provided including automated tools and existing mission applications previously built for reuse.
- Development : a project team performs development. At this step, I&RTS recommends to develop iterative manner as MIL 498 or RUP recommends.
- Submission : a project manager submits completed segments to be tested for the COE compliance.
- Integration : a project manager and DISA integrates segments for validity check with the COE and place the segments on the automated repository.
- Installation : distributes segments through network or compact disks. The installation is automatically performed by supporting tools.

4.4 Mission Applications Development in Pilot Project

After defining a segment, architecture, and development process, we also developed three simple tools representing the tools of the COE listed in I&RTS. They are simple automation tools for installing (`COEInstaller`), verifying (`VerifySeg`), and searching segments (`COEScanCOTS`). We selected two domains, then developed two mission applications from each domain. One is "Aircraft Maintenance Information System (AMIS)" and the other is "Tactical Air Support System (TASS)." The AMIS helps aircraft maintenance people supporting administrative work keeping the aircraft status data such as operational or under maintenance of aircraft. The TASS enables front line warriors to request air support from pilots in airbase post. The TASS usually depends on the AMIS in such a way that aircraft maintenance people provide resources to warriors and pilots, so high interoperability and reusability are required.

We organized three teams of graduate students. One team took the responsibility for supervising project development, and other two teams are for mission application development. Each development team was assigned to build one mission application. Before starting the project, a development team manager should acquire an approval from the supervision team. After that, the supervision team provided the COE that includes the three tools above. After one team completed the AMIS, then the other team started to implement TASS in order. AMIS was built on the basis of client/server system using Javabean and EJB techniques, while TASS was built as a webserver using JSP and EJB. The Fig. 5 shows the screen shot of each mission applications. Finally we performed the compliance level check.[6]

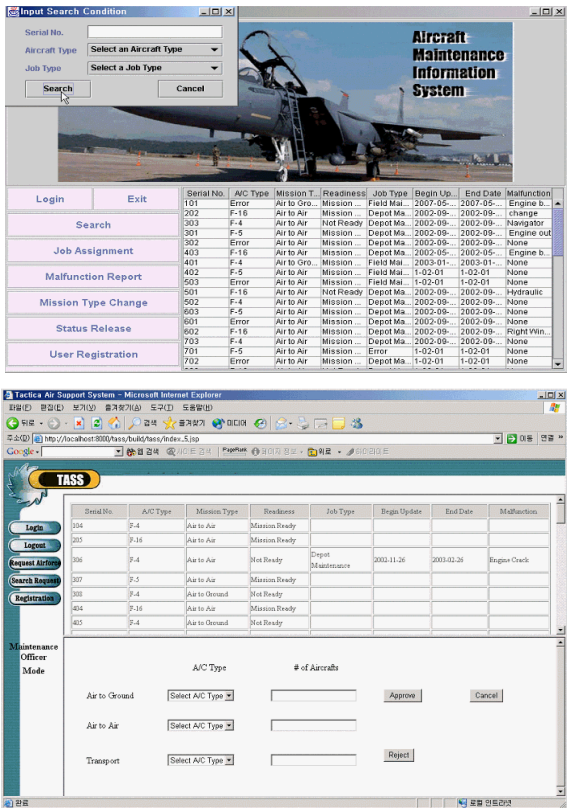

Figure 5. The screen shot of AMIS and TASS

Some problems and difficulties were raised as follows. First, we could not have a clear idea of COTS that I&RTS strongly recommend to include. In some aspect, COTS may be harmful for interoperability and reusability in that any COTS vendor must compete with other rivals. As a result, COTS of a vender usually are not interoperable with the COTS of other vendors. Second, the guideline of I&RTS admits too much room for developers to understand and apply the guideline. During the development, each developer understood the guidelines by his/her own taste and suffered difficulty to reuse. Third, through the iterative development, sometimes developers forgot to update descriptors after revision of segments. To keep consistency between a segment and its descriptor was not an easy task. Finally, some COTS is not so perfect to be interoperable. For ex-

[6]The artifacts we developed can be evaluated as level 4 according to the the checklist in I&RTS, but the details are omitted here for space limit.

ample, Java virtual machine is supposed to provide platform independency, but we had some problems on Linux and Windows2000. Something running on Window2000 is not working on Linux properly.

5 Conclusion

So far, we investigated the availability of Java techniques in realization of the DII COE philosophy. Before the development of mission applications we created some prototype tools which are to help developers of mission applications. We also cover some issues in the DII COE realization, and show certain ways to deal with. The heterogeneity of segments distributed on various types of network needs descriptors to cope with the problems of composition validity check. The collaboration based mechanisms that can be used to deal with inter-segment problem are introduced

The items of the compliance level check in I&RTS actually do not cover the issues of soundness of segment compositions. We are considering to develop a reference model of the COE for compliance level check that may addresses the collaboration issues, and may be helpful for developers who may need a practical example of the COE concept realization. In our work, we missed an important issue on the COE, that is security. Basically, infrastructure of the COE (such as J2EE) is responsible for low level security. We will try to search for practically applicable and more effective security features especially for mission applications.

References

[1] H. Gomma, Software Design Method for Concurrent and Real-Time Systems, *Addison Wesley*, 1993.

[2] W. Tracz and L. Coglianse, "An Outline for a Domain Specific Software Architecture Engineering Process," *Fourth Annual Workshop on Software Reuse*, Reston, VA, 1991.

[3] C. Hofmeister, R. Nord, and D. Soni, Applied Software Architecture, *Addison-Wesley*, 2000.

[4] LTC E. Mettala and M. H. Graham, "The Domain-Specific Software Architecture Program," *Special Report, CMU/SEI-92-SR-9*, 1992.

[5] US DoD, "Defense Information Infrastructure(DII) Common Operating Environment(COE) Integration and Runtime Specification(I&RTS)," Defense Information SA, 1999.

[6] US DoD, "Defense Information Infrastructure(DII) Common Operating Environment(COE) Java Development Guidelines, v 1.0," Defense Information SA, May 2000.

[7] D. Batory, C. Johnson, B. Macdonald, and D. V. Heeder, "Achieving Extensibility Through Product-Lines and Domain-Specific Language: A Case Study," *ACM Trac. on Software Engineering and Methodology*, Vol 11, No. 2, April 2002, pp. 191-214.

[8] Y. Smaragdakis and D. Batory, "Mixin-Based Programming in C++," *GCSE 2000*, 2002, pp. 163-177.

[9] M. Mezini and K. J. Lieberherr, "Adacptive plug-and-play compnets for evolutionary software development," *Proc. of OOPSLA 98*, pp. 163-177, 1998.

[10] J. S. Lee and D. H. Bae, "An Enhanced Role Model for Alleviating the Role-binding Anomaly," *Software-Practice & Experience, John Wiley & Sons*, vol. 32, issue 14, Nov. 2002.

[11] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison Wesley, 1998.

[12] G. Frazier, "The DII COE: An Enterprise Framework," *Science Applications International Corporation (SAIC)* , Oct 2001.

[13] B. H. Liscov, "A Behavioral Notion of Subtyping," *ACM Trac. on Programming Languages and Systems*, vol 16, No 6, Nov 1994, pp. 1811-1841.

[14] R. B. Findler, M. Latendresse, and M. Felleisen, "Behavioral contracts and Behaviral Subtyping," *ESEC/FSE 2001*, 2001.

[15] M. Schrefl and M. Stumptner, "Behavior-consistent specialization of Object life cycles," *ACM Trac. on Software Engineering and Methodology*, Vol. 11, No. 1, Jan 2002, pp. 92-148.

[16] B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, 1988.

[17] R. Rajkumar and M. Gagliardi, "High Availability in the Real-Time Publisher/Subscriber Inter-Process Communication Model," *Proc. of the IEEE Real-Time Systems Symposium Dec. 1996*, IEEE Computer Society Press, 1996.

[18] N. Medvidovic, P. Oreizy, and R. N. Taylor, "Reuse of off-th-shelf components in C2-style architectures," *Proc. of the 1997 symposium on Software Reuseability*, 1997, pp. 190-198.

[19] B. B. Kristensen, "Object-oriented modelling with roles," *Proc. of the 2nd International Conference on Object-Oriented Information Systems*, 1996.