UMM: UNIFIED META-OBJECT MODEL FOR OPEN DISTRIBUTED SYSTEMS

R. R. RAJE
Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 723 W. Michigan Street, SL 280, Indianapolis, IN 46202, USA
E-mail: rraje@cs.iupui.edu

Abstract:- The success of organizations, in the era of interconnected world, will depend upon their ability to create robust and efficient software realizations for open distributed systems. Such systems, due to their nature, will require combining distributed objects (or components) adhering to different models. In this article, we present an overview of a meta-model, which allows a seamless integration of heterogeneous software components.

Key-Words:- Distributed, Objects, Meta-model, Aspects, Collaboration, Service

1 Introduction

In this new century, we will be much more dependent on the advances in the field of Information Technology. This observation is emphasized in the President's Information Technology Advisory Committee (PITAC) report. This report has identified five priorities for research, software being the first one [1]. It also recommends research in component-based software design, and the exploration of protocols, languages, data structures to promote interoperability of applications running current across wide-area networks. This paper addresses some of these challenges, especially as applied to the domain of open distributed systems.

Distributed systems (DS) assume the view that "the network is the computer" and are becoming ubiquitous in many critical domains. These systems are mainly characterized by a presence of heterogeneity, local autonomy over hardware and software resources, built-in openness and a probability of partial failure. Despite the achievements in software engineering, the development of software for large-scale de-centralized systems is still a challenge. As more organizations will be required to design open DS, it will be necessary, for these organizations, to create robust and effective software solutions for DS.

Out of the existing options, the software design of DS can efficiently be achieved by using the principles of distributed-object computing (distributed objects, by their definition, are components and hence, in this article we treat the terms objects and components as synonyms). Under this paradigm, DS are viewed as a collection of a large number of heterogeneous, interconnected,

(possibly) mobile and "smart" components. These components constantly discover one another, offer/utilize services and negotiate the cost and the quality of these services. Such a view not only provides a scalable solution but also presents an uniform access to underlying resources, while hiding their heterogeneity.

Various distributed component models have been proposed by industry and academia. Each have their strengths and weaknesses. However, almost a majority of these have been designed and/or deployed for 'closed' systems, i.e., systems, although distributed in nature, running in a confined setup, such as a large corporation. In contrast, the software realization of open DS will require combining components that adhere to different models, thereby, increasing the complexity of software development. Hence, a comprehensive framework is required to tame this complexity and aid in the design of DS.

Many important issues need to be considered during the design of such a framework, specifically: a) a design of a meta-model for components, b) a multi-level approach for indicating the contracts and associated constraints of the components, c) guidelines for specifying and verifying the quality of components, d) a formal mechanism for precisely describing the computation achieved by each component, e) a set of rules for static and dynamic integration of components, f) mechanisms for evaluating the quality of the resulting component assemblages, g) a methodology for component-based software design, and h) validating this framework by creating proof-of-concept prototypes of distributed applications from different domains. Due to the space constraints, here we only describe the general architecture of the meta-model and briefly discuss related features.

2 Background

Many models have been suggested for distributed-component computing. For the sake of brevity, here, we briefly describe only the prominent ones.

Java-RMI and Jini: The computational model of Java is *write once and run anywhere*. The basic model of Java is incapable of initiating computations in a remote address space. To overcome this deficiency, Sun has released a mechanism, called RMI (Remote Method Invocation), which allows a programmer to invoke methods on specified remote-objects [2,3]. RMI is a language-centric model, where Java acts as the interface and implementational language. The Jini [4] architecture aims at replacing the notion of peripherals and applications with that of network-available services and clients that use those services. There are three components of the Jini architecture – a) infrastructure, b) programming model and c) services. Jini is built on Java and RMI.

Although, Jini is a correct step in the direction of providing a unified view to distributed computing, it does assume a homogeneous view, i.e., everything is Java, a view which may not be pragmatic in open systems.

CORBA: CORBA interfaces are specified in IDL (Interface Definition Language) [5,3]. The implementations of the objects can be written in any language, as long as there exists a proper mapping between IDL and the implementation language. A client object, residing on one machine, may request that an operation be performed on an object whose implementation resides on a different machine. This is accomplished by making use of the Object Request Broker (ORB), which is responsible for finding the object implementation, preparing the object implementation to receive the request, and communicating the data comprising the request. In addition, CORBA provides IIOP (Internet Inter-Object Protocol), which allows interoperability across the networks. CORBA supports many facilities and services and allows dynamic invocations via DII (Dynamic Invocation Interface). However, it does not allow complete object-to-object connectivity (due to a presence of the broker) and its introspection abilities are minimal (as offered by DII, without concrete notions of quality of service).

DCOM: DCOM [7,3] is Microsoft's answer to the distributed-object world. It is based upon their earlier model, COM. It is also a language independent model (interfaces are specified in MIDL). However, unlike CORBA, it is not a platform or operating system independent model. It does offer some level of introspection capabilities, as it allows a set of functions bound to a certain object that implements them, and which may be queried by a user employing the built-in QueryInterface function.

Web Object Model and DOM: The popularity of the WWW and its limitations have been well analyzed by many researchers [8]. They have argued that an attempt is necessary to amalgamate the web model and the distributed-object model [9]. Manola [8,9] has proposed a 'web object model', that allows the construction of object-like facilities that enable the development of powerful Web applications. The Document Object Model (DOM) proposed by W3C [9,10] provides a scripting interface to the document and defines how to manipulate the objects found in a HTML or XML document. Both these aim at integrating the principles of the Web and distributed-objects, but lack the richness of a meta-model.

Pragmatic Object Web: Pragmatic object web [11] is defined as the emerging synthesis of multiple object models. It is aimed at creating a hybrid object system. They describe different application projects which, in fact, reaffirm the motivation of our model. However, they do incorporate the traditional high-performance computing as the back-end, thereby constraining

their approach. Also, they do not address factors such as precise specification of computation, user preferences, aggregations, market-based economic models, which are issues that form key components of our proposed model.

In addition to these models, there are many projects, such as Hadas [6], Infospheres [12], Legion [13], Globus [14], etc., which aim at developing infrastructures for open computing. A large percentage of them are created for traditional high-performance applications and are based on one of the existing (or a variation of an existing) distributed-object model. Hence, we have not described these projects here.

In recent past, there have been many projects, such as Agent TCL [15], Aglets [16], JATLite [17], etc., based on the principles of intelligent agents. Distributed (and mobile agents) are considered as one of the promising alternatives to develop open systems. Although agents are a powerful concept, we feel that they are at a higher level of abstraction (than objects). Also, many of the agent projects/frameworks use one or the other existing distributed-object model at the low-level and hence, we are not considering the agent model during the discussion of our proposed model. However, in our model, we have incorporated a few concepts from the domain of agents.

3 Meta Model for Objects

3.1 Why a Meta-model?

Components, by their definition, are independent of the language implementation, tools and the execution environment. However, as described earlier, many different models exist for the development and deployment of components in DS. Also, to a large extent these models were designed for geographically dispersed, but closely controlled systems. Hence, they will fall short when applied to future systems, which will be inherently open and distributed. Each model assumes only its own presence, which contradicts the local autonomy principle of DS. Even if any of the existing models becomes universally accepted, it will require the re-engineering of many systems, which may be technically sound, but not economically viable. Also, many of these models lack comprehensiveness as they do not consider concepts such as quality and economic frameworks. An open distributed system will, certainly, contain components adhering to different models. These components will have to interact with each other to facilitate the software realization of DS. In recent past, scripting languages such as Truce, Perl, Tcl, JavaScript, and Python have been used to 'glue' heterogeneous components. Such an approach is suitable for rapid prototyping; but, it does not contain the complete power of a rich meta-model. Hence, there is a need for a comprehensive meta-model, which will seamlessly

encompass existing models. The design of any meta-model should extract useful features from existing approaches, but, reorient, integrate and enhance them, and incorporate innovative concepts to increase its comprehensiveness.

The first step in the creation of a meta-model is to investigate the existing models with an in-depth study of their architectures, and identifying strengths and weaknesses by developing prototype systems. Many of our past and present projects, such as ARMI [18] and D-SIFTER [19], are aimed at exploring and experimenting with existing models. These experiments have provided us with a sufficient basis, from the perspectives of both architecture and application design, for the creation of the meta-model.

Below we describe a Unified Meta-object Model, which seamlessly encompasses existing and future models, thereby providing a solution for the software development of open distributed systems.

3.2 Unified Meta-object Model (UMM):

The three parts of the UMM are *object, service* and *collaboration.* The innovative aspects of the UMM are in the structure of these parts and their inter-relations.

Object:

In UMM, objects are autonomous entities, whose implementations are non-uniform, i.e., each object adheres to some distributed-object model and there is no notion of either a centralized controller or a unified implementational framework. Each object has a state, an identity and a behavior. Thus, all objects have well-defined interfaces and private implementations. In addition, each object in UMM has three aspects: 1) *Computational Aspect*, 2) *Cooperative Aspect*, and 3) *Auxiliary Aspect.*

Computational Aspect: The computational aspect reflects the task(s) carried out by each object. It in-turn depends upon: a) the objective(s) of the task, b) the techniques used to achieve these objectives, and c) the precise specification of the functionality offered by the object. In DS, objects must be able to 'understand' the functionality of other objects. Thus, each object in UMM supports the concept of introspection, by which an object will precisely describe its service to other inquiring objects. There are various alternatives for an object to indicate its computation – ranging from simple text to formal descriptions. Both these extremes have advantages and drawbacks. The use of formal techniques will certainly alienate the ambiguity, but can add to an overhead (as formal methods are harder to understand and, thus, remain a forte of a smaller number of highly talented developers). The UMM takes a

mixed approach to indicate the computational aspect of an object – a simple textual part, called *inherent attributes* and a formal precise part, called *functional attributes*. The inherent attributes are represented by a tuple of the form $<$ *id, creator, object model, version, date*$>$. The name of each component of the tuple is self-explanatory of its significance. The inherent attributes will be specified by the developer of that object.

The functional part is formal and indicates the precise nature of the computation, its associated contracts and the level(s) of the quality of the service offered by the object. In [20] an argument is made in the favor of multi-level contracts for objects (components). They have classified the contracts into four levels – syntactic, behavioral, concurrency and quality of service. UMM integrates this multi-level contract concept into the functional part of the computational aspect. In an open system, each component will be offering a service (more about it is discussed later) and hence, the level related to the quality of service is especially critical in UMM. The level of the quality depends upon factors such as, the algorithm used, the execution model, resources required, time, precision and classes of the results obtained. UMM makes an attempt at quantifying the quality of service by creating an appropriate vocabulary and providing multiple levels of quality, which could be negotiated by the components involved in an interaction. The functional part is specified in a formal fashion by using the principles of denotational semantics. This part will also be specified by the creator of the component.

Cooperative Aspect: In UMM, objects do not exist in isolation. They are always in the process of cooperating with each other. This cooperation may be task-based (to achieve a certain task in DS, an object may associate with other objects, delegate sub-tasks to them and coordinate their activities) or greed-based (an object may offer its services only to accumulate compensation for its owner). The cooperative aspect depends on many factors: detection of other objects, cost of service, inter-object negotiations, images of collaborators, aggregations, duration, mode, and quality. More details about this aspect is mentioned in the collaboration part of the UMM.

Auxiliary Aspect: In addition to computation and communication, mobility, security, and fault tolerance are the necessary features of an open DS. The auxiliary aspect of an object will address these features. In UMM, each object can be potentially mobile. The developer of the object will decide whether it is mobile or not and for what duration it will stick to a specific role (stationary or mobile). The mobility of the object will be shown as a 'mobility attribute' (a notion similar to the inherent attribute). If an object is mobile, then the mobility attribute will contain the necessary information, such as its implementation details and required execution environment. Similarly, secu-

rity in any global system is a critical issue. It is evident on many levels: i) intra-object ('how secure is each object?'), ii) inter-object ('how secure is an inter-object collaboration?'), and iii) resource ('how secure are the resources to which objects will have accesses?'). The security attribute of an object will contain the necessary information about its security features. As DS are prone to frequent failures, full and partial, fault tolerance is critical in these systems. Similar to mobility and security, each object, in UMM, contains fault-tolerant attributes in its auxiliary aspect.

As local autonomy is inherent in open DS, forcing every object developer to abide by certain rigid rules, although attractive, is doomed to fail. UMM tackles the issue of non-uniformity with two special categories of objects – *head-hunter* and *translator*. These objects are responsible for allowing the seamless integration of different object models and sustaining a cooperation among heterogeneous (adhering to different models) objects. Although, head-hunter and translator objects are special categories of objects, they also have all the three aspects in them.

`Head-hunter Objects`: The tasks of these objects are to detect the presence of new objects, offer memberships to them, register their functionalities, communicate and attempt match-making between service producers and consumers, while demanding their compensations (more about this is discussed later). A head-hunter object is analogous to a broker, or a binder or a Trader in other models, with one important difference – a broker is passive, i.e., the onus of registration is on the object and not on the broker. Thus, in a brokered model, an object (either a service provider or a seeker) initiates the registration process, as opposed to a head-hunter, which is active – it discovers other objects and makes an attempt to register them with itself. During the registration process, each object will inform the head hunter about its inherent, functional, mobility and security attributes. The head hunter object will use this information during the process of matching. An object may be registered with multiple head-hunters. Once a head-hunter creates a match between the supplier and seeker objects, it receives its compensation and no longer interferes in the communication between the matched pair. Head-hunters may cooperate with each other, (as these are also objects with cooperative aspects), in order to cater to a larger number of seeker/supplier objects. The functionality of the head hunter objects makes it necessary for them to communicate with objects belonging to any model, implying that the cooperative aspect of head hunter objects be universal, i.e. able to facilitate collaboration with any object. Once a head hunter object has created a seeker-supplier pair, the concerned objects, in this pair, must be able to establish a direct communication. Considering the heterogeneous nature of the objects, it is conceivable that the seeker and the

provider be compliant to different models, thus, creating a need for a *translator* object.

`Translator Objects:` A translator object acts as a mediator between two objects adhering to different models. Thus, the cooperative aspect of a translator object has two parts - one each for the two models, to which the communicating objects adhere. The computational aspect of the translator object will indicate the two models it is proficient in. Like other objects, translator objects will register with the head hunters and while doing so, they will indicate their specialization (which models they can bridge efficiently). During a request from a seeker, the head hunter object will not only search for a provider, but it will also supply the necessary details of a translator, if deemed necessary.

Service:

The concept of a service is the next part of the UMM. A service could be an intensive computational effort or an access to underlying resources. Irrespective of the nature of the service, objects offer them with an intention of accumulating compensation for their owners. The notion of service is a prerequisite for the concept of collaboration. In DS, it is natural to expect that each seeker have several choices for obtaining a specific service. Thus, each object, in addition to indicating its functionality, must be able to specify the cost and quality of the service offered. A seeker object will consider these factors while deciding on its collaborator.

The nature of the service offered by each object is dependent upon the computation performed by that object. In addition to the algorithm used, expected computational effort and resources required, the cost of each service will be decided by the motivation of the owner and the dynamics of demand and supply. In a dynamic environment costs must always be accompanied by the duration for which the costs are valid. As the system dynamics undergo constant changes, the methodologies used to fix the cost of a service will evolve as time progresses, thereby creating a need to indicate the time sensitiveness of the cost. The quality of service will be an indication given by an object, on behalf of its owner, about its confidence to carry out the required services in spite of the constantly changing execution environment and a possibility of partial failures. The techniques used to determine the cost, the time-validity and the quality of a service will depend upon the tasks carried out by the object and the objectives of its owner and will involve principles of distributed decision making.

The service offered by each object crosscuts its there aspects: a) the syn-

tactic and quality features are reflected as a part of the computational aspect, b) the behavior ('what exactly is this service offering and what state changes will take place as a result of the execution of this service?') and the synchronization ('how would the service be affected in a concurrent scenario, i.e., will be the service affected if more than one clients access it concurrently and what kind of safe guards would be needed to ensure a specific-level of service?') are incorporated in the cooperative part, and c) any special features ('is this service fault-tolerant?') are indicated in the auxiliary aspect. Each object will publish these attributes and will export them to the concerned seekers.

Collaboration:

The third and the final part of UMM is collaboration. Collaboration is a need for DS and a consequence of service. Collaboration is affected by many factors: detection, negotiations, images, aggregations, duration, mode and trust. The detection of collaborators, in UMM, is handled by the head hunter and translator objects (as described above). In an open environment there are many choices for a specific service and hence, each object must be able to negotiate with other objects. The notion of a large sea of objects meshes well with market-driven economic models. Objects demanding services must compensate the service providers (including head-hunter and translator objects). Once an inter-object communication is established, the associated objects will indulge in negotiations. This concept is similar to the one suggested (from the domain of intelligent agents) by Genesereth and Ketchpel [21]. Selecting a collaborator depends upon many factors ranging from trivial ones like the cost and quality of service to more sophisticated ones like maintaining images of different service providers. For example, depending upon the past experiences (how well and promptly was the task completed?), a seeker may choose to select a more expensive provider over a cheaper one. These images will be updated continuously, thereby, indicating the dynamic nature of the distributed applications.

Also, as indicated earlier, the costs of the services are time and market-dynamics dependent. Thus, each object should have an ability to negotiate with the others so as to adjust the costs and quality factors. Once an object has selected a collaborator, UMM then allows two modes of obtaining service: a) buying – a copy of the provider will be forever owned by the seeker, and b) leasing – a service is provided at an agreed cost for a certain amount of time and can be renewed (the idea is identical to the one used in Jini).

Just like the service component, the collaboration component of UMM is also spread over all three aspects: a) the inherent attributes ('who?') and the

functional attributes ('what?') are a part of the computational attribute, b) the detection of other objects, inter-object negotiations, communication modes, images of the collaborators and selection are reflected in the cooperative aspect, and c) any special features of the object and/or of its collaborators are present in the auxiliary aspect of the object.

Thus, each object in UMM is formally represented as a 3-tuple: <CA, CoA, AA>, where CA = computational attribute, CoA = cooperative attribute and AA = auxiliary attribute. Each component of the tuple is further divided into sub-tuples, the size of each is variable. The interface of each object comprises of CA, CoA and AA. As each object in UMM must contain all the three aspects, preferably each class (objects are assumed to be instances of classes) in UMM will preferably implement Universal Interface (UI). An UI consists of: a) common features related to CA, b) common features related to CoA, c) common features related to AA and d) reflexive features (as an open environment will decree the use of dynamic detection of unknown objects). In addition, the UI will consists of the necessary mechanisms for low-level communication and collaboration, thus, relieving the communication burden from the application developer.

Anant – An Environment Based on UMM

At present, we are in the process of creating *Anant* (meaning *endless*), an environment based on UMM. Anant is being developed using Java, due to its inherent advantages. As that work is in the preliminary stages, here we describe only a few features of Anant.

For the benefits of a new environment to reach a large audience, it is necessary to provide an easy-to-use and effective interface to it. Thus, Anant will have a personalized interface, which will lessen the burden of the user. We are proposing a notion of a *personalized object (PO)* – another category of the general object in UMM. The tasks of a PO will be to accept queries from the user, contact the head hunters and translators (if needed), create an ensemble of objects necessary to achieve the task (as indicated in the query), coordinate the activities of all the constituent objects, collect the results, compose them and present them to the user. The user will specify the queries using the formal representations of all the aspects of the objects which they want PO to seek on their behalf. The results (which themselves may be objects with associated execution environments) will be 'personalized' by the PO based on the user preferences. The user will specify his/her preferences to the PO through a use of a standard layout language like XML. The notion of PO is similar to the personalized agents from the domain of AI, but more specialized due to the use

of formal specification of the aspects as the 'communication language' between the user and the PO, rather than a natural language or an agent-language.

Testing and Validation

Any new environment must be evaluated and validated by developing prototype systems, experimenting with them, gathering feedback from them and incorporating the experiences gained into the environment to enhance its capabilities. In addition, these prototypes will provide the necessary means to compare and contrast with existing alternatives. Currently, we are involved in many inter-disciplinary projects, such as information filtering, which will ideal test-beds to validate UMM and Anant. As the development of Anant is in its preliminary stages, here we have not described any specific application scenario involving UMM. However, the applicability of UMM to many different domains, which involve distributed components scattered over open and interconnected system, is very obvious.

4 Significance and Conclusions

This paper addresses some of the key issues that will have to be resolved as the operations of society's enterprises expand ever more globally. Monolithic computing systems will no longer be capable of dealing with the geographical extension and computational complexity such operations will require. The response to this will be the creation of systems that automatically adjust with a minimum of human intervention to the changing requirements of the enterprise and the supporting open DS.

The meta-model, UMM, by its nature, does not decree the use of a specific implementational model. Also, it does allow a seamless integration of existing objects, with a minimum reorganization. The basic components of UMM, object, service and collaboration, themselves are not new. However, their internal structures, associated contracts, QoS, and explicit integration is innovative. These also provide an illusion of uniform and simplified access to underlying heterogeneous computational and network resources. An incorporation of a new model, in UMM, will only need the design and development of appropriate translator objects and modification of head hunter objects. Such an approach will not only tackle the model-heterogeneity but also address the scalability of the model and the environment.

References

1. President's Information Technology Advisory Committee. *Information*

Technology Research: Investing in Our Future, 1999.
2. Sun Microsystems, Inc. *Java^{TM} Remote Method Invocation Specification*, 1996.
3. Orfali R, and Harkey, D. *Client/Server Programming with JAVA and CORBA*. John Wiley & Sons, Inc., 1997.
4. Waldo, J. The Jini Architecture for Network-centric Computing. *Communications of ACM*, 42(7):76–82, July 1999.
5. Siegel, J. *CORBA Fundamentals and Programming*. John Wiley & Sons, Inc., 1996.
6. Israel, B. and Kaiser, G. Coordinating Distributed Components Over the Internet. *IEEE Internet Computing*, pages 83–86, 2(2), 1998.
7. Microsoft Corporation. *DCOM Specifications, www.microsoft.com/oledev/olecom*, 1998.
8. Manola, F. Towards a Object Model. *www.objs.com/OSA/wom.htm*, 1998.
9. Manola, F. Technologies for a Web Object Model. *IEEE Internet Computing*, 3(1):38–47, January-February 1999.
10. Wood, L. Programming the Web: The W3C DOM Specification. *IEEE Internet Computing*, 3(1):48–54, January-February 1999.
11. Fox, G. The Document Object Model Universal Access Other Objects CORBA XML Jini JavaScript etc. *www.npac.syr.edu/users/gcf/msrcobjectsapril99*, 1999.
12. Infospheres Project. *www.infospheres.caltech.edu*, 1998.
13. Legion Project, www.cs.virginia.edu/~legion, 1999.
14. Globus Project. *www.globus.org*, 2000.
15. Kotz, D., Gray, R. *et al,*. Agent TCL: Targetting the Needs of Mobile Computers. *IEEE Internet Computing*, pages 58–67, 1(4), 1997.
16. Aglet Project. *www.trl.ibm.co.jp/aglets*, 2000.
17. JATLite Project. *java.stanford.edu*, 2000.
18. Raje, R. *et al,*. An Asynchronous Remote Method Invocation (ARMI) Mechanism for Java. *Concurrency: Practice and Experience*, 9(11), 1207-1211 (November 1997), 1997.
19. Raje, R. *et al,*. A Bidding Mechanism for Web-Based Agents Involved in Information Classification. *WWW Journal* , 1(1998):155–165, 1998.
20. Beugnard, A., Jezequel, J. *et al,*. Making Components Contract Aware. *IEEE Computer*, 32(7):38–45, July 1999.
21. Genesereth, M. and Ketchpel, S. Software Agents. *Communications of the ACM*, 37(7), 48-53, 1994.