METAOASIS: An Architecture Description Language for Open Distributed Systems Modeling

Ángeles Lorenzo, Isidro Ramos, José A. Carsí
Dpto. de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
www.upv.es

Jorge Ferrer
Tecsidel
Information Technologies
www.tecsidel.es

Summary: This article presents the new expressive facilities that have been added to the formal OO specification language OASIS in order to obtain an Architecture Description Language (ADL). OASIS thus becomes a powerful tool for modeling Open Distributed Systems and also for automatic generation of prototypes using a model compiler approach. The reflexive mechanisms provided by OASIS and the functionality of its meta-level extend the structural, functional, and dynamic aspects obtaining, thus a powerful ADL framework.

Key Words: Architecture Description Language, Open Distributed Systems, Reflection, Component and connector

1 Introduction

One of the trends in the technological evolution of software is directed towards the creation of Open Distributed Systems (ODS) software and consequently, it is also a challenge that has been raised in the domain of Software Engineering (SE). The new technology's rapid evolution is due to the increasingly demanding requirements of the business sector, needing solutions for complex information systems, the globalization of business transactions, and the dynamic changes within organizations.

Software Engineering’s commitment to the construction of Quality Software must not be forgotten when faced with the demand for rapid solutions. Nevertheless to make this feasible, SE must respond to the new systems features (Concurrency, Evolution, and Distribution) and propose new paradigms capturing the requirements that have been defined for this type of systems in an adequate way.

The Object Oriented Approach has been the SE mainstay for closed systems but there are many questions to respond for open environments such as the lack of an overall vision of the system, the difficulty of guaranteeing security and confidentiality in data transmission, the heterogeneity of the components and platforms, etc. [12].

According to the new trends, the technology based on the object oriented model has evolved towards Client/Server architectures, distributed DB and towards GroupWare technology; changing the Object Oriented Paradigm towards a new computational model: Distributed Objects. Distributed objects are suitable for the creation of flexible and distributed systems because the data and business logic are encapsulated within them, independent of the system places, thus improving the distribution’s granularity.

A distributed object can be defined as an intelligent “globule” which is capable of residing in any of the system’s places, more specifically; they are pieces of an independent code which clients can access by the invocation of their methods. A new concept the COMPONENT emerges as well as the system’s overall structure details. This approach is currently being undertaken by a new discipline: Software Architecture, and this focus is especially suitable for the definition and development of complex and distributed systems.

In this context, we present MetaOasis as a model and a language able to capture in an elegant and precise way at a formal specification level the architectural aspect of the distributed systems. Our document is structured in the following way: after the introduction in section 2, we present the concept of Software Architecture as integrative of the different tendencies that seek answers to the new software necessities. Next, how the industrial standards provide a solution for the
distributed objects through business objects. In the last part of this section the reflection idea is reached as the formal base to demonstrate the dynamic capacities of Architectural Definition Languages. In section 3 the reflexive model of OASIS is shown; over on this we will support our model of components that will allowing to construct distributed system architectures.

2 State of the Art

2.1 Software Architecture (SA)

SA is a high level representation of the structure of a system describing the parts that integrate it, the interactions among them, the patterns that supervise its composition and the restrictions when these patterns are applied [7]. Thus, the parts that integrate the system are the components and the interaction that take place among them are performed via connectors.

A component is a compositional unit of software having a set of interfaces, which can be developed, acquired, incorporated and included into the system in an independent way with respect to other components in space and time. These interfaces are the parts of the components that can provide and require services. As a result, we can identify a component as a Distributed Object.

The programming paradigm based on Distributed Objects follows an architectural perspective that promotes the development and construction of Reusable Business Objects. Reusability is understood as its adaptation capability for new requirements and its versatility allowing them to be plugged into contexts that are different from the ones for which they were designed.

Thus, this new concept is going to change the traditional idea of software development and it is going to result not only in innovations in the implementation methods but also significant changes in the specification and design phases.

2.2 Industrial Technologies

The demand for distributed objects is a fact and due to this, a wide range of technical solutions are available although none of them are definitive. It is true that for years the software industry has never stopped debating about objects and their reuse, but until the moment no standards exist, there have been only tentative proposals.

A.- Standards

In spite of the trade war that exists among manufacturers, two standards stand out above the rest: OpenDoc/CORBA and OLE/DCOM. For both, the market’s components must be compliant with the following properties:

- It must be a marketable item: an autonomous piece of software that can be acquired in the open market.
- It can be combined with other components to form complete applications.
- It can be used in unforeseen combinations.
- It must possess a clearly defined interface.
- It is an inter-operative object: it can be invoked as an object between directional spaces. As a result, it is a software item that is independent from the system.

B.-Business Objects

The distributed objects are by definition components. As Fig 1 shows the structure of a distributed component can be defined in three levels:
domain objects. In the higher abstraction level we have business components/objects implementing the semantics of the tasks.

2.3 Reflection

During the last two decades, computational reflection has been an active research topic especially since its introduction in the objects-oriented field [9].

Computational reflection is defined as the system’s capability to think about and act upon itself. This consists of the application of “levels separation” criteria with the purpose of designing self-modifying systems with a theoretical foundation.

Reflection has been used in a wide range of applications and the most common results are an increased transparency in communication, distribution, and mobility as well as powerful solutions related to self-adaptation and intelligent behavior modeling.

Reflection is not a concept that is restricted to research labs but it is also currently gaining recognition in the market. Since the end of the 80’s, the Smalltalk programming language has already contemplated the use of meta-classes in its definition. At present, it is a property of the components that are sold by CORBA and DCOM, which supply the treatment of meta-data and introspection; features which Sun [11] already include in the Java 1.1 version by means of API and Microsoft via the IDispatch interface [2].

3 OASIS: A Conceptual frame for Distribution Modeling

3.1 Oasis Reflection

The OASIS object model has been extended in a reflective manner, introducing the meta-class concept, thereby obtaining, a model with behavioral reflection that allows designers to define, validate and modify software systems. The Transaction-Frame Logic (TFL) language was used to specify and implement the OASIS meta-class in a formal way.

An OASIS schema can be reified as a meta-object state. This meta-object is composed of other meta-objects, one for each class of the schema. The state of a meta-object defines the structure and the behavior of all the possible instances of the associated class. This state can be changed through the services of the meta-object.

Each meta-object has a double view (Fig 3). On the one hand it is an object with a local state, and offers services to define the set of properties that constitute the template that will share all the objects that it creates. On the other hand it is a class, and can create and destroy its instances.

The meta-class concept in OASIS is defined as a compound class that gathers all the
characteristics of the OO model. Schematically, the meta-class will be an abstract editor of OASIS classes, which are created by means of its services. OASIS provides three units of design which are: the primitive classes, the elemental classes and the complex classes (aggregation class, permanentSpecialization_class and temporalSpecialization_class). The meta-class allows the user to construct classes of any of these three categories.

Meta-class services allow us to define and change the characteristics of the classes (through the modification of the state of the meta-objects). The method associated with each service defines the semantics associated with the change it represents, including object evolution in order to be compliant with the new definition. Invariants that must hold are checked as service preconditions or class constraints.

The OASIS meta-class is the complex meta-object reifying the templates of meta-classes (the OASIS model), which allow for the creation of meta-objects. The state of these meta-objects defines the characteristics of primitive, elementary and complex classes. The OASIS meta-class offers as unique services the creation and destruction events of instances. Its population is composed of the set of meta-objects having as state the set of defined schemas, and its template is the composition of the set of meta-classes templates (Fig 4).

In order to close the cycle and to avoid having a potentially infinite hierarchy of meta-classes, the reflective closure of meta-class is constructed so that the OASIS meta-class is said to be an instance of itself. At the moment, and because it has so been designated, the structure of the OASIS meta-class is invariable and defines the OASIS model.

3.2 References to the Client and the Server Architecture

Our starting point deals with the questions that were left unsolved in [8] when a change was considered in the communication mechanism among the objects to accommodate it to the Software Architecture scheme.

An answer is being sought in the reflexive possibilities defined in the OASIS language. We follow the formalized approaches presented in [5] where the introduction of reflexive concepts in the architectural description allows the dynamism and the evolution of the systems to be specified in a flexible and uniform way. For this is necessary to solve questions such as:

- What is a component in Oasis?
- How can it be constructed in a homogeneous way and how can it be used in different contexts?
- To provide a new diagram for the communication among objects based on the
definition of connectors that manage the occurrence relations between the clients’ actions and the server’s actions.

- To solve the question, for the sake of maintaining the homogeneity, if the connectors are components.
- To deal with the questions of the instantiation and reuse of a connector.
- The connector dynamics and evolution.

In previous works related to reflection, the meta-components are used in order to reify the links that are defined in the base level. In MÁRMOL [5], the connectors are meta-spaces and are identified as any other component. There are also authors that have defined reflexive connectors [1] [6] but in almost all the cases they are implementations that generate their code starting from an architectural description.

The proposal provided by J. McAffer [10] and improved by W. Cazzola [4] is the one taken for the expansion of the OASIS reflexive model.

McAffer defines a group of seven elements in his reflexive CodA Architecture that are present in the meta-level of all the objects and that manage the base level interactions: Send, Accept, Queue, Receive, Protocol, Execution and State.

This proposal can be interpreted inside our model as Fig 5 shows:

- The service request is made by the object through its class and it reifies the request in the schema. (1)
- The acceptance of the service is the response in the Meta-Class if the message is correctly formulated and constitutes a part of the services that can be found in the templates of the diagram’s classes. (2 and 3)
- The queuing process can be supplied on the three levels: the Object, the Class, and the Meta-Class, depending on the type of request:
  - The object that has requested the services specifies the request’s server object: Object, Method, Arguments. (9)
  - A service is requested from an object of another class: Class, Method, Arguments. (3)
  - The request is launched to the diagram waiting to be solved by some event that is present in the class templates that belongs to the schema. (2)

According to this diagram, the request for service will be handled by:

- A specific object or,
- By any of the objects that belong to the class to which the request is addressed, or

---

Fig 5: Distributed Architecture based on OASIS reflexive model
• By any of the objects that belong to one class able to provide the service requested. The corresponding queues supply the subsequent asynchronous request that must be handled, in accordance with the policy implemented.

4 Concluding Remarks

In this article the enhanced OASIS 3.0 model is presented as a model for the Description of Software Architectures. The objective is to give an answer to the Specification of Distributed Open Systems problem. Our proposal will allow the automatic generation of prototypes to validate and deals with non-functional requirements. The work by Carsí [3] and Letelier [8] define an appropriate framework to extend the OASIS language and fulfil the lacks of many LDA’s concerning their capacity to capture the dynamics of the systems and the non-functional requirements.

The expressiveness of this language is enriched with three important elements: the definition of obligation and prohibition processes, the incorporation of the client perspective and the extension of its model with a metal-level. This frame offers an appropriate and complete formal support:
• To specify hierarchies in a natural way,
• To deal with non functional requirements and
• To capture the dynamic aspects of the system.

However, we must avoid that the high abstraction level of the language move away from its final objective: to specify Distributed Objects Conceptual Models assuring the finer granularity of the distribution. To cope with this problem Model's expressiveness needs to be completed with elements that allow us to build a total Software Architecture. Our approach will provide a uniform reference frame to define all this problem aspects.

The works in development are:
• To implement a tool generating in automatic way, prototypes of Open Distributed Systems.

References