Mokum for Correctness by Design in Relation to MDA*

Reind van de Riet
Emeritus, Dept. of Computer Science
Vrije Universiteit in Amsterdam

Abstract. The use of the Mokum system for correctly designing Information Systems and how these designs can automatically be translated into implementations, is the theme of this paper. Similarities and differences with tools from the field of Model Driven Engineering/Architecture, such as UML, will be demonstrated.

Keywords: Model Driven Architecture, Knowledge base systems.

1 Introduction

In the old days one believed in the possibility to design correct Information Systems (IS) and to implement them. Currently, one is more realistic, and is concerned with tools to find errors and to remedy these systems.

The Mokum system is from the old days; the acronym stands for Manipulating Objects with Knowledge and Understanding in Mokum (Mokum is another name for Amsterdam). We want to show in this paper what its principles are, and why we believe that these principles are still valid, because one can automatically generate an implementation from a design, proven to be correct. The following principles are being used:

1. **the IS encompasses all players in the field**, not just the data. So in a hospital IS, these are the doctors, the nurses, the financial people, etc. In an IS for a garage, not only cars to be repaired or sold are represented but also the technicians, the selling people, their bosses, the managers and the administrative employees. The ground for this principle is that we want to be able to prove correctness of access and usage of software tools.
2. **design and implementation are one and cannot be considered independently**.
3. **all players and data to be defined are ordinary objects**, in the realm of Object-Oriented programming.

During design phase and implementation phase the type tree is of crucial importance. The type tree is a tree in which all the types of the objects to be considered are defined. It is used to define access from one object to another, and also to pieces of software.

---

* MDA is the trademark of the Object Management Group OMG; MDE is free.
This last criterium is translated in the rule that an object can only execute a piece of software in the form of a trigger. All triggers are defined within a type(s) of the object and react upon a message sent by another object (or itself). Dependent on the state an object is in it will/can react. Suppose it is asleep, then only a timer message can awake it. Suppose it is waiting for a reaction of another object, it may be only susceptible to a message of this other object.

We will see that it is (in principle) easy and straightforward to automatically translate in Mokum a design, in the form of WorkFlow diagrams, on a high level into the definitions of these triggers. That means that with a correct design comes a correct implementation. For more on WorkFlow diagrams see [1].

How do we achieve this goal of having a correct design? There are two reasons: due to the simple principles upon which access control is based, it is possible to use simple tools (in Prolog) to prove that access for security and privacy is guaranteed; the second is that we use semantic knowledge, in the form of Ontologies, to validate the design. This validation does not have the power of a proof of correctness, but it helps to ascertain the high quality of a design. For reading about Mokum in Cyberspace and Security & Privacy issues see [11].

2 The Mokum System

As said above, Mokum is an object-oriented system, with special notions to handle access control. (See also [4-16]). There is a type hierarchy, with the usual inheritance of properties. Each type has a number of attributes, which themselves may be of simple nature, such as text and integers, or of a newly defined type, such as for the father of a person. In addition a type has several states and a script, in which triggers are defined which react on incoming messages, by changing states, values of attributes, and sending messages to (other) objects. There is the notion of time, so that timers can be set. The is_a relation defines the inheritance of properties. If T is_a S then type T has all the properties of type S, not only attributes but also states and script. So an object being an instance of type T is automatically an instance of type S. All types are is_a related to the special type thing. The only property of thing is that it defines unique identifiers for its instances, the so-called object identifiers: OID. All types can be put in a tree, the type tree, structured according to the is_a relation as hierarchical principle. In the diagram for the type tree the is_a relation is indicated by means of an arrow with a closed head. The is_a relation is transitive.

What makes Mokum special is the notion of collection and its keeper. In order to make it possible to define access control, these notions have been introduced. Suppose that an object, like a doctor in a hospital system, needs single access to a set of other objects, of a certain type, in this case type patient, we make this object keeper of that set. So in the example case: we make the doctor keeper of the collection of (his or her) patients, to which only he or she has access. To indicate the relation between keeper and collection, there is a special relation between types the coll_of relation, indicated as follows: suppose the keeper has type T and the collection is a set of objects of type S, then in the type tree there is a special arrow from T to S, an arrow with open head. The coll_of relation is non-transitive. In a case where there is only one