This paper presents a computation model and its programming language, \textit{A'UM,*} as a result of our pursuit of high parallelism and high expressivity for the development of a large scale software. By basing it on streams and integrating it with objects and relations, \textit{A'UM} realizes an elegant model, natural representation and efficient execution, all at once, that have never been done by any other approaches.

\textbf{Keywords:} Concurrent Programming Language, Stream, Object-oriented

§1 Introduction

In general, the larger a problem is, the harder it is to solve, exponentially proportional to the problem size. Our goal is to realize a computation system in which large or complicated problems can be solved quickly in terms of the total amount of time spent for the designing, programming, debugging, maintenance and extension. To achieve this goal, the system should satisfy two kinds of requirements at once: one is to offer natural and flexible user interfaces, including languages and design/program/debugging environments; the other is to realize an efficient implementation. Neither natural user interfaces nor efficient implementations can exist without any sound computation model.

First, we need a computation model that can extract maximum parallelism from the problem and load minimum overhead to the architecture, and provide high level abstractions. Furthermore, we don’t want a model such that

\textit{* \textit{A'UM} is a Japanese word, derived from a Sanskrit “ahum” consisting of \textit{Ah} and \textit{Um}, which implies the beginning and the end, and open voice and a close voice, and expiration and inspiration. This name was given to symbolize stream communication which is the basic notion of this language.}
mixes several paradigms, since it is hard to formalize, understand and realize. At first, the model must be simple and uniform with a single fundamental notion. Abstractions should be built up based on this notion.

In this paper, we present a computation model for completely distributed systems and a language realizing the model, A'UM, which is characterized by the following three features:

- Stream-based computation,
- Object-oriented abstractions, and
- Relational representation.

§2 Background
There have been many theories, models and languages proposed for concurrent systems.21)

2.1 Poset-Based Modelling
Concurrent systems are those in which there can exist independent events. Between these independent events, there is no precedence or constraint on which one should happen earlier or later.

While a totally ordered set (or chain) is a set in which any pair of elements in given some order, a partially ordered set (poset) is a set which may contain elements which have no order. Since posets reflect the concurrent situation naturally, concurrent systems have been formalized based on posets, while sequential systems have been formalized based on chains. The most notable work at this side is Scott's lattice theory42) and information system theory.41)

Ref.41) gave a formal language in which a system is represented as an algebraic formula of posets (strictly pomsets) of events. The representation is quite descriptive for programming.

There are other algebraic approaches, called process algebras.37,38,50,10,51) A process is defined to be a set of ports. Communication between processes is basically synchronous and global time and space are assumed to exist. Two systems are said to be equivalent if their sets of linearized events are the same. When a new system is added to some system, the prover has to re-generate all possible chains from all events in the existing system and in the newly added system. This would require an extravagant amount of computation. In other words, models based on the total ordering in global time and space are weak in modularization.

In contrast, the sheaf-theoretical model39) has high modularity. It is assumed that a system has a location and any activity takes place in some location. Communication between systems takes place in the intersection of their locations. Two systems are said to be equivalent if the set of intersections of locations in one system is equivalent to that in the other. The notion is clear