MODELO PARA EL DISEÑO DE SISTEMAS ORIENTADO A OBJETO CON PROCESAMIENTO PARALELO

Angela S. Chikhani C.*
Universidad Simón Bolívar.

Hugo Segovia
Universidad Central de Venezuela.

RESUMEN

Se presenta un modelo para el diseño de sistemas orientado a objeto, dirigido a ambientes de desarrollo en arquitecturas de computadores con procesamiento paralelo. Se define formalmente el modelo, el cual esta fundamentado en los trabajos de Coad/Yourdon. La finalidad de éste estudio es ofrecer un modelo para el diseño de sistemas, con notaciones familiares, que permite minimizar los problemas que se presentan al analista y al programador, en la programación para computadores con procesamiento paralelo. Adicionalmente, se evalúa el enfoque y la sintaxis de los lenguajes Pool y Mentat con relación al modelo desarrollado.

Palabras Claves.

Procesamiento Paralelo - Orientado a Objeto - Diseño de Sistemas.

1. Introducción.

En la actualidad, la utilización del procesamiento paralelo en las aplicaciones comerciales, trae como consecuencia el desarrollo de nuevos lenguajes de programación y de nuevas arquitecturas de computadores. Entre los nuevos enfoques desarrollados para la programación destinada a computadores con procesamiento paralelo, se pueden citar: Orientado a Objeto, Funcional y Lógico [1]. Cada uno de estos aporta ventajas comparativas, tanto en el desarrollo de aplicaciones como en la programación.

Grimshaw, Strayer y Narayan [2,3], plantean la existencia de dos grandes problemas en la programación en arquitecturas con procesamiento paralelo. Primero, el desarrollo de programas

* Sede del Litoral. Dpto. de Tecnología Industrial. Municipio Vargas. Dtto. Federal. Venezuela.
Tel.: (031)722911 al 18 ext 220 y 211- Fax: (031)722313 - e-mail:chikhani@skynet.usb.ve

para arquitecturas con procesamiento paralelo es complejo, debido a que es necesario un correcto manejo del medio ambiente de programación a nivel de comunicación, sincronización y planificación de procesos independientes. Segundo, el código implementado para una arquitectura no es, en la mayoría de los casos, portable a otras arquitecturas.

En éste trabajo se presenta un modelo para el diseño de sistemas orientado a objeto enfocado a arquitecturas de computadores con procesamiento paralelo, basado por un lado en la metodología de diseño orientada a objeto utilizada por Coad/Yourdon [5,6], y por otro en los lineamientos de los lenguajes de programación orientada o objeto para arquitectura de computadores con procesamiento paralelo (POOPP), publicados por diversos autores [1,3,7]. El principal aporte de éste estudio, es ofrecer un modelo para el diseño de sistemas con notaciones familiares, de fácil implementación mediante los lenguajes POOPP, que minimiza los problemas que representa, para el analista y el programador, la programación en arquitecturas con procesamiento paralelo.

2. Fundamentos del modelo.

En éste trabajo, se propone un modelo que permite reducir los problemas planteados anteriormente, ofreciendo al analista/programador la posibilidad de simplificar los aspectos de dependencia, comunicación, sincronización y planificación de datos, en el desarrollo de aplicaciones.

El modelo es similar al modelo de diseño orientado a objeto descrito por Coad/Yourdon [6], en la definición de atributos y servicios, identificación de estructuras, dominios, clases y objetos. También considera de forma similar, las actividades de diseño de los componentes dominio del problema, interacción humana y componente manejador de datos. Sin embargo, el modelo difiere en tres de los principios para el manejo de la complejidad, estos son los principios de abstracción, encapsulación y comunicación. Así como también en el diseño del componente manejador de tareas. Por otra parte, la existencia o no del paralelismo dentro de un nodo de la red, no se considera en la notación del modelo, esto se debe a la doctrina de los lenguajes POOPP [1,3,7], para los cuales, la encapsulación del paralelismo dentro de un nodo es responsabilidad del compilador. El modelo contempla, al igual que los lenguajes de POOPP, la comunicación

sincrónica y asincrónica. A continuación se describen cada uno de los aspectos en que difiere el modelo al modelo de Coad/Yourdon y los aportes de los lenguajes POOPP.

El principio de abstracción permite ocultar temporalmente los datos y procedimientos. Si se considera la programación en arquitecturas de computadores con procesamiento paralelo, se debe incluir la abstracción del paralelismo dentro de un objeto, es decir, la abstracción del paralelismo que puede ocurrir en la ejecución de los servicios de un objeto. El modelo propuesto, permite la abstracción en la ejecución dentro de un objeto, ofreciendo la posibilidad de ejecutar el objeto en cualquiera de las modalidades de procesamiento bien sea paralela o secuencial. Por otra parte, el ocultamiento de los datos del objeto previene el acceso directo de los datos privados, simplificando el proceso de control para los datos del objeto.

Una de las características principales en el principio de encapsulación, es la abstracción de datos y la comunicación con mensajes. Para considerar el procesamiento paralelo, es necesario extender la noción de abstracción de datos y comunicación, adicionando la encapsulación del paralelismo. Un aspecto importante de los lenguajes POOPP, es la transparencia en la encapsulación del paralelismo dentro y entre llamadas a procedimientos [2]. En el modelo objeto de estudio, se consideran dos tipos de encapsulación: intraobjeto e interobjeto. Se denomina encapsulación del paralelismo intraobjeto al ocultamiento de la implementación, secuencial o paralela, de un procedimiento dentro de un objeto. Mientras, la encapsulación del paralelismo interobjeto, se refiere a la explotación del paralelismo en las llamadas a los procedimientos de un objeto de manera transparente al programador. El modelo contempla la encapsulación del paralelismo intraobjeto e interobjeto y pueden además combinarse.

En los lenguajes POOPP [1,3,7], la invocación a un procedimiento dentro de un objeto no bloquea la ejecución en paralelo del resto de los procedimientos, siempre que lo permita la dependencia de datos. Para estos lenguajes, cada objeto tiene una actividad independiente, diferentes objetos se ejecutan en paralelo e interactuan explícitamente en el envío y respuesta de mensajes. Esta comunicación puede ser de forma sincrónica o asincrónica. El modelo propuesto en este trabajo, permite al analista definir el tipo de comunicación a ser empleada en la aplicación a desarrollar. La notación para la conexión de mensajes en el modelo sería la siguiente:

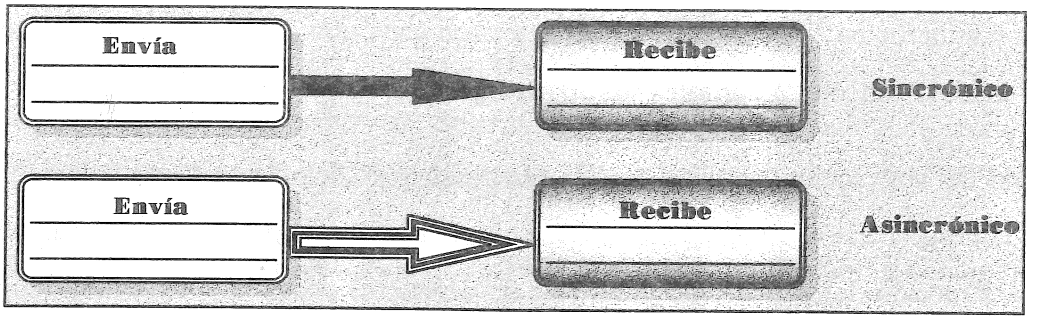

Figura 1. Notación para el envío de mensajes.

Una observación a la notación de Coad/Yourdon [6], es que el modelo que ellos describen es completamente estático y no refleja adecuadamente la dinámica de los servicios. A continuación se presenta un diagrama del comportamiento de los servicios con el que el envío de mensajes sincrónico y asincrónico.

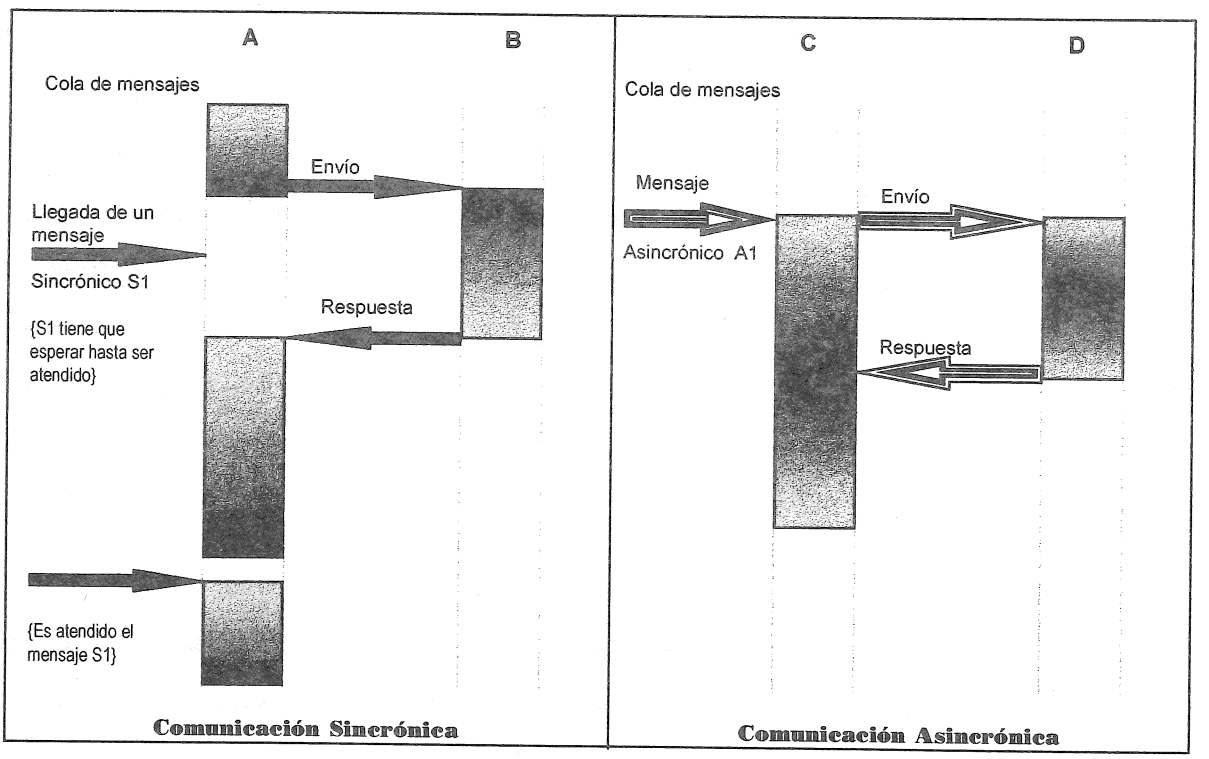

Figura 2. Comportamiento de los servicio en el envío de mensajes.

Por otra parte, en la ejecución con procesamiento paralelo, el nivel más alto se aplica a trabajos y programas, mientras que el siguiente nivel se aplica a procedimientos o tareas dentro del mismo programa, esto supone la descomposición de un programa en múltiples tareas. El tercer nivel trata

de explotar la concurrencia entre múltiples instrucciones y con frecuencia, se realiza un análisis de dependencia de datos para revelar paralelismo entre instrucciones. Finalmente, se puede desear disponer de operaciones más rápidas y concurrentes dentro de cada instrucción.

Si se considera, por una parte el modelo de multicomponentes descrito por Coad/Yourdon [6] y por otro lado el segundo nivel de procesamiento paralelo, es necesario definir para el modelo que aquí se presenta un componente manejador de tareas con procesamiento paralelo (CMTPP). La figura 3 muestra la notación del CMTPP.

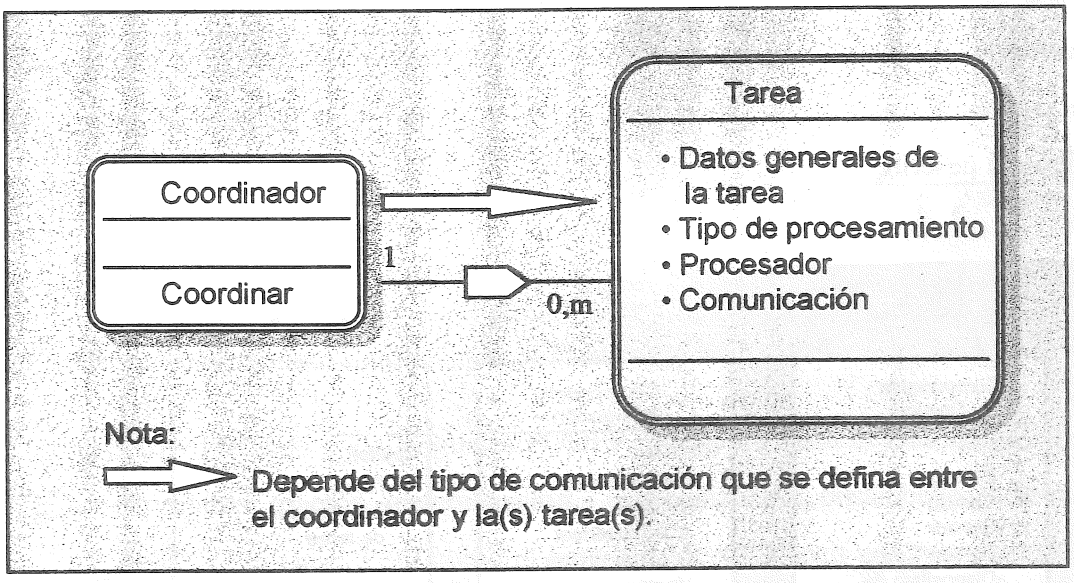

Figura 3. Notación para el CMTPP.

En la figura 3 se debe destacar que el aspecto datos generales de la tarea esta definido por: nombre de la tarea, descripción, prioridad, servicios incluidos, coordinador, vía de comunicación. Adicionando a la estructura original, toda la información relevante referente al segundo nivel del procesamiento paralelo, es decir, tipo de procesamiento, procesador y tipo de comunicación entre tareas.

2.1.Ejemplo. Sistema Monitor de Sensores.

Aplicando el modelo propuesto al ejemplo presentado por Coad/Yourdon [6] y expandiendo los lineamientos del problema original para así considerar el diseño de tareas con procesamiento

paralelo, considerando una arquitectura paralela donde cada procesador esta equipado con una memoria local en la cual pueden ser almacenados los datos de los objetos y la cola de mensajes. De esta forma, el analista/programador puede utilizar ubicación indirecta para especificar donde quiere inicializar el nuevo objeto o dejar que el sistema escoja el procesador que inicializará el objeto. Por ejemplo, puede especificar que desea que el nuevo objeto se encuentre en el mismo procesador o que no se encuentre ubicado en el mismo procesador de un objeto específico.

El sistema monitor de sensores, consiste de un sistema que permite controlar sensores que detectan situaciones de alarma en edificios y reporta condiciones criticas. El sistema mantiene la pista de cada sensor con la dirección del edificio donde esta instalado y el número de emergencia asociado. Además el sistema contiene datos importantes como la fecha para reparar los sensores y el estatus de cada alarma. A continuación, la figura 4 muestra los componentes del sistema y la figura 5 describe la CMTPP.

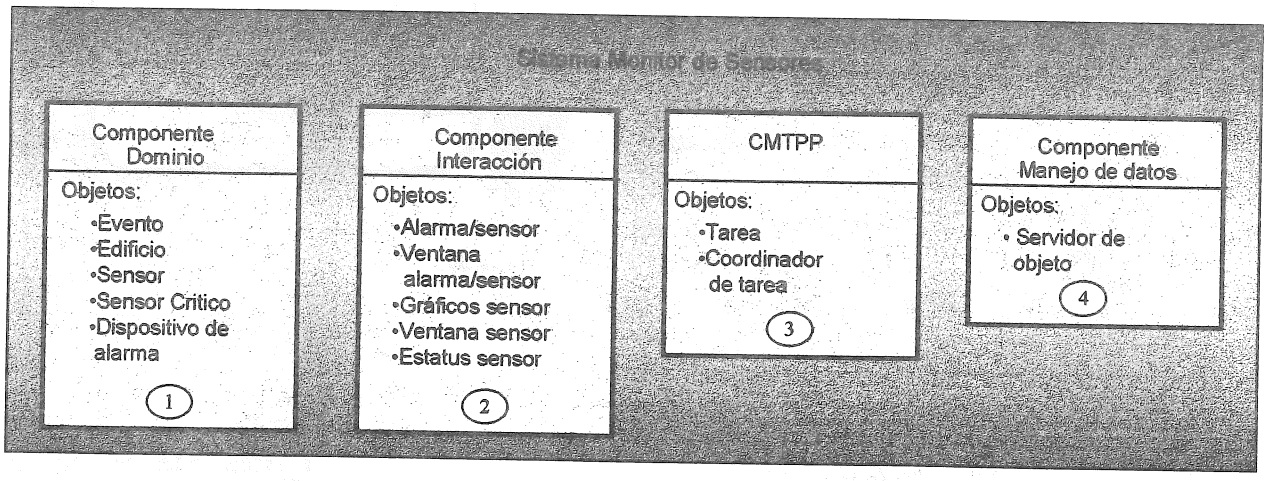

Figura 4. Componentes del Sistema Monitor de Sensores

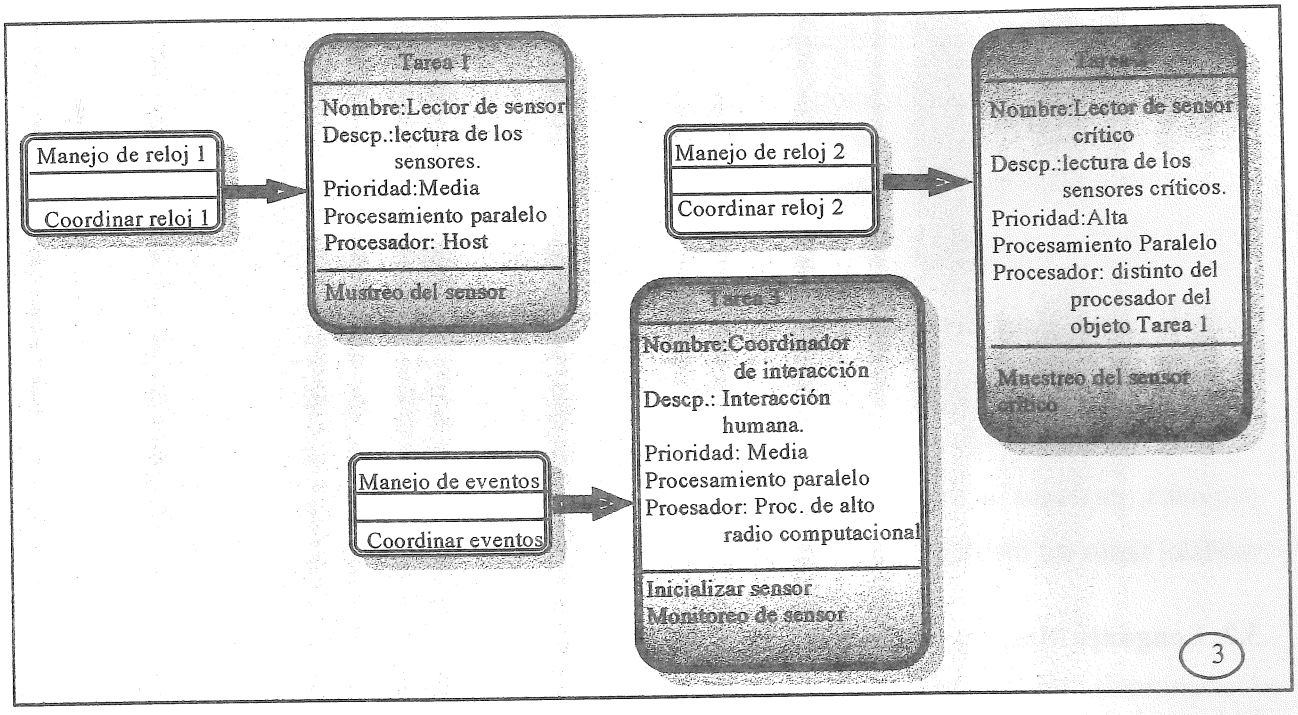

Figura 5. Descripción de CMTPP.

3. Correspondencia entre el modelo y los lenguajes POOPP.

La correspondencia entre el modelo y los lenguajes POOPP, puede ser vista evaluando el soporte en aspectos tales como: clases, objetos, generalización - especialización, ensamblaje, atributos y servicios. Tanto la familia de lenguajes Pool como el lenguaje Mentat tienen una correspondencia casi directa con el modelo, lo que le permite al analista/programador pasar del diseño al lenguaje de una forma relativamente sencilla.

3.1. Lenguajes Pool.

Según los diseñadores de la familia de lenguajes Pool, Annot y Haan [1], para estos, las clases no son consideradas para ser objetos ellas mismas. Sin embargo la correspondencia entre el modelo y el lenguaje es casi directa, es decir, el lenguaje maneja clases y objetos al igual que el modelo. Además el modelo permite apreciar todas las características relacionadas con éste aspecto.

La familia de lenguajes Pool contempla la definición de clases dentro de una unidad y el manejo de apuntadores a objetos dentro de objetos, es decir, permite la anidación de objetos y esto se denota

en el modelo como estructura de ensamblaje. Además los lenguajes Pool soportan herencia simple y múltiple, es decir, generalización y especialización.

En cuanto a las variables, estas están accesibles a toda la unidad del programa, así como a los métodos definidos en una clase. Las variables se denotan en el modelo como atributos.

Los lenguajes Pool definen métodos para objetos y rutinas para clases. Los objetos, para comunicarse, emplean el pase de mensajes sincrónico o asincrónico y una acción de comunicación entre dos objetos puede ser interna en un nodo o externa, en el caso que el mensaje sea partido en paquetes, enviados a través de la red y ensamblados nuevamente en el nodo destino. En el modelo, tanto los métodos como las rutinas, reciben el nombre de servicios.

3.2. Lenguaje Mentat.

El lenguaje Mentat desarrollado por Andrew S. Grimshaw [3], se presenta como una extensión del lenguaje C++. Este lenguaje define clases y objetos de forma similar que el modelo, lo que representa una correspondencia directa. Para éste lenguaje, los objetos pueden ser declarados estática o dinámicamente y las clases poseen distintos tipos de construcciones, es decir, un miembro de una clase puede ser declarado privado, protegido o público. Además se consideran dos tipos de clases: la clase persistente y la clase regular.

Al igual que la familia de lenguajes Pool, Mentat soporta herencia simple y múltiple. Además, Mentat soporta el principio de ensamblaje del modelo, bien sea, en la anidación de objetos o en el enlace de apuntadores implementado en el control entre objetos. Por otra parte, las variables pueden estar acotadas o no acotadas.

La llamada a una función miembro de un objeto, es sintácticamente la misma que en los objetos de C++. No obstante, semánticamente existen dos diferencias importantes, la función miembro son siempre llamadas por valor y la invocación no bloquea la ejecución. Una función miembro se denota en el modelo como servicios.

4. Conclusiones.

El presente trabajo ofrece un modelo para el diseño de sistemas basado en el análisis de Coad/Yourdon. Entre sus ventajas están su notación familiar, su fácil implantación en los lenguajes POOPP y su contribución a la portabilidad de aplicaciones en ambientes de arquitecturas de computadores con procesamiento paralelo.

El principio de encapsulación que contempla el modelo, se puede aplicar en aquellos desarrollos en los cuales no se afecte el grado de paralelismo.

El modelo propuesto puede ser implementado fácilmente mediante los lenguajes de programación orientada a objeto, Pool y Mentat.

El modelo planteado simplifica los aspectos de dependencia, comunicación, sincronización y planificación de datos en el desarrollo de aplicaciones.

Este modelo permite reducir el tiempo que emplea el analista/programador en el desarrollo de nuevas aplicaciones en ambientes de computación con procesamiento paralelo.

5. Referencias.

[1].Treleaven University College London, "Parallel Computers Object-Oriented, Functional, Logic.", P.C.Treleaven, WILEY.1992.

[2].A.S. Grimshaw, W. T .Strayer, P. Narayan, "Dynamic, Object-Oriented Parallel Processing", IEEE Parallel & Distributed Technology, May.1993, pp. 33-46.

[3].A.S. Grimshaw., "Easy to Use Object-Oriented Parallel Programming with Mentat" Computer, Vol.26, Nro.5, May 1993, pp. 39-51.

[4].Collier, William W., "Reasoning about Parallel Architectures", Prentice-Hall International Editions,. 1992.

[5].Coad and E.Yourdon, "Object-Oriented Analysis", Prentice Hall, Inc, New Jersey, 1990.

[6].Coad and E.Yourdon, "Object-Oriented Design", Prentice Hall, Inc, New Jersey, 1991.

[7].Bershad, E.D. Lazowska, and H. M. Levy, "Presto: A System for Object-Oriented Parallel Programming", Software:Practice and Experience, Vol.18, Nro. 8, Aug 1988, pp.713-732.