VirD-GM: Introducing a Modelling and Execution Environment for the Distributed Geometric Machine Machine*

Vanessa S. Fonseca, Renata H. S. Reiser,
Adenauer C. Yamin, Antonio C. R. Costa, Mauricio L. Pilla
Catholic University of Pelotas, Informatics School
Pos-Graduate Programme in Computer Science
Rua Feliz da Cunha, 412, Centro, Pelotas, Brazil CEP 96.010-000
{vandag, reiser, adenauer, rocha, pilla}@ucpel.tche.br

Abstract

This work describes the integration of the Geometric Machine model (GM) with the execution environment for pervasive and grid computing EXEHDA. The model is responsible for the logical structure of the processes and the related graphic expressions may be constructed in the interface of the Visual Programming Environment of the Geometric Machine model (VPE-GM), while the middleware EXEHDA carries out its parallel and distributed execution. The resulting environment, named VirD-GM, is a simple and reliable framework where the performance of several parallel programming tasks can be analyzed in an uniform way. The visual environment VPE-GM handles the entire workflow, from modelling to execution of algorithms of scientific computations, and it is especially useful for partial, concurrent and non-deterministic programs. The validation of graphic representations together with the control of process constructors and memory configurations are obtained in the process editor and memory editor, respectively. This means that graphic representation free from semantic errors in the relationship in the GM process constructors and their memory configurations can be exported to the execution module in a higher abstraction level.

Keywords: Geometric Machine Model, Parallel Computations, Distributed Computations, Distributed Computing Environment.

Resumen

Este trabalho descreve a integração do ambiente de programação visual para o modelo de Máquina Geométrica (MG) com o ambiente de execução para computação pervasiva denominado EXEHDA. O primeiro, refere-se a estrutura lógica das expressões gráficas representando os processos do modelo MG e o segundo, reponsabiliza-se pela correspondente execução paralela e distribuída. Com base nesta integração, o ambiente de execução, denominado VirD-GM, implementado sobre o middleware EXEHDA, constitui-se numa ferramenta simples e confiável onde o desempenho de diversas tarefas da programação paralela podem ser analizadas de forma uniforme, para modelagem de algoritmos da computação científica, especialmente algoritmos parciais, concorrentes e não-determinísticos. O ambiente de desenvolvimento, denominado VPE-GM, é capaz de prover a validação das representações gráficas, incluindo o controle e configuração da memória. Significando que a construção gráfica mapeando a relaçao entre construtores de processos do modelo GM e as correspondentes configurações de memórias estão livre de erros semânticos e podem ser exportada para o módulo de execução no midlleware EXEHDA.

Palabras clave: Modelo de Máquina Geométrica, Computação Paralela e Distribuída, Ambiente de Programação Distribuída.

*This work has been partially supported by the Brazilian funding agency FAPERGS.

1 Introduction

The development and evolution of parallel and distributed computing is highlighted from the different and comprehensive related areas (*High Performance Computing*, *Grid Computing*, *Context-Aware Computing*, *Mobile Computing*, *Pervasive Computing*). These areas are responsible for the interaction among distinguished computational devices connected to global network. Recently, the consolidation of this area enabled dynamic resources sharing and pervasive environments in order to support physical and logical information mobility according to the user's profile or application. The demand for new and powerful computational resources to provide better performance, in a simpler but accurate way, has been increasing. Despite being a priority to the development of new information technologies and current applications, many factors which contribute significantly to supply this demand are committed.

In this work, we are concerned about the gap between theoretical models and the technological development. We present a formal model and related abstractions to support semantic interpretations for parallel and distributed computing.

The main objetive is the integration of the Geometric Machine model (GM) with the execution environment for pervasive and grid computing EXEHDA. The model is responsible for the logical structure of the processes and the related graphic expressions may be constructed in the interface of the Visual Programming Environment of the Geometric Machine model (VPE-GM), while the middleware EXEHDA carries out its parallel and distributed execution. The resulting environment, named VirD-GM, is a simple and reliable framework where the performance of several parallel programming tasks can be analyzed in an uniform way. The visual environment VPE-GM handles the entire workflow, from modelling to execution of algorithms of scientific computations, and it is especially useful for partial, concurrent and non-deterministic programs.

In addition, mathematical structures and logical approach in the GM model provide concepts and (visual or/and textual) languages to capture the essence of any parallel and non-deterministic system, in algorithmic and descriptive terms. Many challenging issues are connected to the design and specification of efficient parallel implementation in different architectures.

The Geometric Machine (GM) [6] is a model for construction and semantic interpretation of parallel and non-deterministic computations applied over array structures with shared-memory, performed by multiple processes distributed over a geometric space. In the construction procedure of such model, an algebraic and structural induction is considered, the former was concerned with finite character of the basic objects called elementary processes and the latter is related to the inclusion relation regarding the partial processes in the ordered construction of the GM model.

This paper investigates the main aspects of the GM model related to the construction of a visual programming environment (VPE-GM), from the formalization of the GM model and specification of a visual programming language for the model [10]. The implementation of such environment aims to support the specifications of the language $\mathcal{VL}(\mathbb{D}_\infty)$ [5]. A bi-dimensional semantic interpretation explores alternatives to simplify the task of designing parallel and non-deterministic algorithms.

The methodology of development of VPE-GM framework deals with graphic transformations applied over basic graphic objects of the environment, which are accessed from the interactive interfaces of memory and process editors. This visual approach is compatible with the inductive construction of the process domain in the GM model and makes the comprehension of recursive processes easier.

Then, the development of an integrated solution using the adaptive, services-oriented middleware EXEHDA (*Execution Environment for High Distributed Applications*) [16] as an execution environment is considered. The resulting framework has been constructed to be also a didactic tool, providing an environment where parallel and distributed algorithms for scientific computation can be validated, simulated and executed.

We start in Section 2 with basic concepts of the formal model and its denotation in the correspondent visual language, showing how traditional textual programming language techniques are combined with the specification of a visual language to implement the visual programming environment. The status implementation is commented in Section 3. Then, the parallel abstractions of the GM model and the related execution environment over which distributed and parallel computations can be improved in their real performance is introduced in Section 4. For that, a review of the EXEHDA middleware and how it manages the constraints imposed by processes from the VPE-GM are discussed in Section 5. Finally, related works and concluding remarks are presented in Section 6.

2 Basic concepts of GM model

The Geometric Machine model is an abstract machine, with infinite memory and constant memory access time, that can be used to analyze the logic and domain-theoretical structures of parallel and non-deterministic algorithms for

scientific computations [7, 8, 9].

The GM memory supports a coherence space of states and is conceived as an enumerable set of stored cells that are labelled by points of a geometric space. In such memory, deterministic machine states are modelled as functions from memory positions to values. Non-deterministic machine states are modelled as families of deterministic machine states (with singletons modelling deterministic states) and computation tests are modelled as functions from memory positions to boolean tests.

Based on the concept of coherence spaces, the ordered structure of the GM model is given by a Coherence Space of Processes, denoted by $\mathbb{D}_\infty$. Over this domain, it is possible to obtain interpretation for (possibly partial, recursive, infinite) processes, operating on arrays of data stored in the GM memory. The GM model gives interpretation for deterministic process constructions, including two types of parallelism – *temporal parallelism*, related to the inductive temporal behavior of processes, and *spatial parallelism*, with processes operating in a distributed approach of the GM model.

The model also provides interpretation for the non-deterministic computations and applies the exponential operators of the coherence spaces in the interpretation of the functional space. Algebraic process constructors are defined by the functors in the $\mathcal{C}osp\mathcal{L}in$ category of the coherence spaces and linear functions [4].

The ordered structure of this model is constructed by levels, starting from the coherence space of the elementary processes to the domain $\mathbb{D}_\infty$, see Figure1. Interpretations are step-wise built following Scott's methodology [12]. Generalizing, each level is identified by a subspace $\mathbb{D}_n$, which reconstructs all the objects from the level before, preserving their properties and relations, and drives the construction of the new objects. The relationship between the levels is expressed by linear functions, called *embedding* and *projection* functions, interpreting constructors and destructors of processes, respectively.

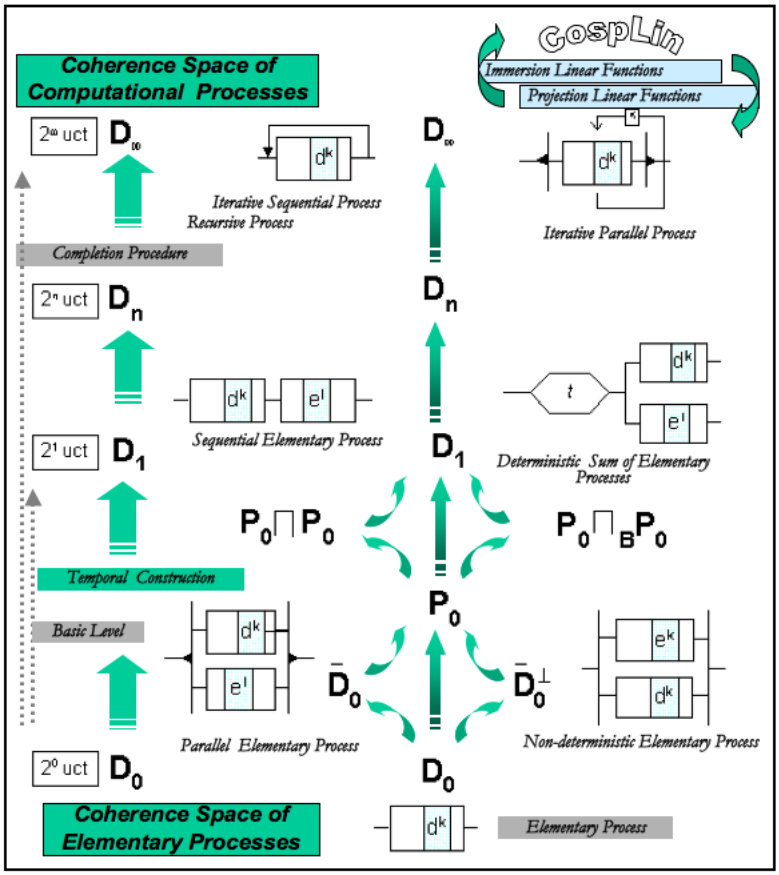

Figure 1: Specifying a parallel process in $\mathcal{VL}(\mathbb{D}_\infty)$

The most basic notion of the GM model is that of *elementary processes* (processes that modify single memory positions in a single unit of computational time, uct). The coherence relation among such processes (coherent subsets) models the admissibility of parallelism. It essentially says that two processes can be performed in parallel if they do

not conflict, i.e., if they write distinct subsets of memory positions. In the dual construction, the incoherence relation models the condition for *non-determinism*, namely, the conflict of memory accesses. Induced by the flat domain modelling of the geometric space, the memory position information on the domain of elementary processes can be lifted to the coherent sets of the constructed domains by a position-function, formally stated in [6].

The completion procedure guarantees the existence of the least fixed point of the recursive equation that generalizes the definition of $\mathbb{D}_n$, the coherence space $\mathbb{D}_\infty$, defined by infinite composition of linear functions. The process representation as objects in $\mathbb{D}_\infty$ is not intuitive, but it allows the construction of textual ($\mathcal{L}(\mathbb{D}_\infty)$) and visual ($\mathcal{VL}(\mathbb{D}_\infty)$) languages.

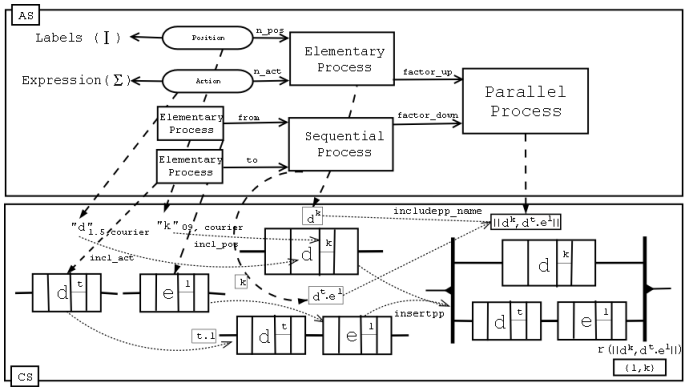

Figure 2: Specifying a parallel process in $\mathcal{VL}(\mathbb{D}_\infty)$

3 Visual approach for the GM model

In the visual approach for the GM model, we consider the alphabet and grammar of the visual programming language $\mathcal{VL}(\mathbb{D}_\infty)$ to manipulate visual information. The visual grammar provides a way to generate all diagrams that constitute the expressions of that visual language, giving an inductive view of the visual language. Two syntactic levels are determined, an *abstract* and a *concrete graphical* level. The former is not concern with the same details as the later. This means that it does not take into account the choice of icons, symbols, and geometric details (size, position) of objects. In particular, disregarding geometric constraints on the abstract syntactic level makes its manipulations by graph transformations simple and powerful. Therefore, it can be used to support applications as well as the future formal semantic definition of the language. In the next subsections, we consider the approach introduced in [1] to describe the syntaxes for these two syntactic levels and to define the fundamental structure of the VPE-GM framework.

3.1 Syntax of the $\mathcal{VL}(\mathbb{D}_\infty)$ language

In the specification of the $\mathcal{VL}(\mathbb{D}_\infty)$ language, typed graphs represent processes interpreted in the GM model, and graph transformations are used to model process constructors. A *process-object* is given by two disjoint sets: *nodes* (visualized as rectangles) and directed *arcs* (visualized as arrows) from a source node to a target node. Every process-object is typed over a type graph, which may be labelled by attributes (visualized as rounded rectangles) that attach information. The position and format of all instances occurring in a diagram depend on graphical restrictions, represented by dotted arrows. In addition, an *attributed arc* connects an attribute node with its current value. Process-attributes may be specified by abstract data types.

In the following examples of symbols in the specification of the $\mathcal{VL}(\mathbb{D}_\infty)$-alphabet, I and ω are enumerable sets of labels (memory positions) and indexes (control of interactions), respectively; Σ represents the set of expressions of the textual language $\mathcal{L}(\mathbb{D}_\infty)$.

We start with the specification of the elementary process $\mathtt{d}^{\mathtt{k}}$ in $\mathcal{L}(\mathbb{D}_\infty)$, as shown in Figure 2. Such process performs the operation $\mathtt{d}$ on the cell with position $\mathtt{k}$ in the machine memory. In the abstract level, the process-object representing an elementary process has two attribute symbols, *Position* and *Action*, (n_act, $action\_name$, n_pos, $position\_name$). In this case, *Position* is the attribution of the position type, which is identified by an element of the I label set. An *Action* is the attribution of an action type, identified by an element in the Σ expression set. In the concrete syntax, these attribution links indicate constraints on the format of the text (size, font, color). Constraint links ($incl\_pos$, $incl\_act$) force the name of the elementary process to be placed in the center of the diagram.

The parallel product is a binary process operator given as a symbol of $\mathcal{VL}(\mathbb{D}_\infty)-$alphabet represented by two process-objects together with the arcs *factor_up* and *factor_down*. These arcs are the connection links that indicate the first and the second factors of concurrent processes in the concrete syntax. Graphical constraints are represented by the inclusion functions $includepp\_name$ and $insertpp$, which help to relate parallel processes to the diagram layout. Figure 2 shows an instance of a parallel product between an elementary process and a sequential process, denoted by $\mathtt{d}^{\mathtt{k}}$ and $\mathtt{d}^{\mathtt{t}} \cdot \mathtt{e}^{\mathtt{l}}$ in the textual language $\mathcal{L}(\mathbb{D}_\infty)$.

At the bottom right side of the concrete syntax, memory positions related to the execution of that parallel process are presented, given by the image set of the position-function:

$$\Upsilon(\| \mathtt{d}^{\mathtt{k}}, \mathtt{d}^{\mathtt{t}} \cdot \mathtt{e}^{\mathtt{l}} \|) = \{k, l, t\} \subseteq I, \text{ with } k \neq l \neq t.$$

The construction rules in $\mathcal{VL}(\mathbb{D}_\infty)-$grammar are structured in two steps, called *preparation* and *realization*. Each one is modelled by a subgraph transformation defined over the $\mathcal{VL}(\mathbb{D}_\infty)-$vocabulary in three steps (Figure 3): *(i)* the partial objects are introduced; (ii) the memory-position constraint indicated by rectangles named $\Upsilon(p) \cap \Upsilon(p) = \emptyset$ and $\Upsilon(p) \cap \Upsilon(p) \neq \emptyset$ are related to concurrent and conflict relations, respectively; (iii) the process constructor is applied over the partial objects obtained in the preparation step.

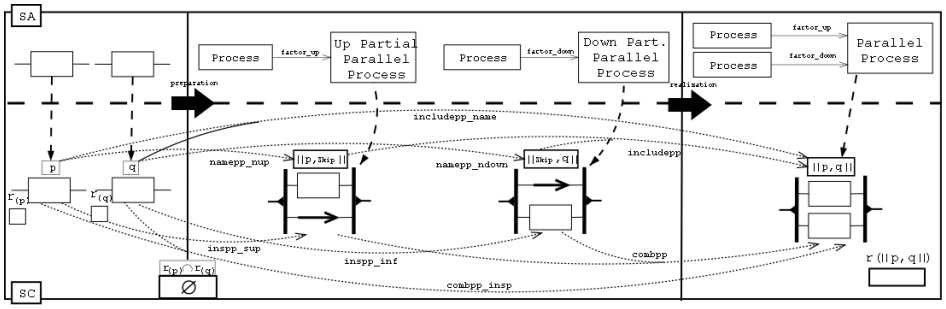

Figure 3: Preparation and realization steps of a parallel product.

3.2 Visual programming in the GM model

The construction of a visual programming environment for the GM model aims to improve the access, manipulation and understanding of the information obtained by performance of parallel computations interpreted in the GM model, including the interface modelling its processes, text and memory states. As it usually happens in the traditional programming languages, it is also possible to iterate sequential and concurrent constructions. Its modular structure is simple, consistent, and adds relevant aspects to visual simulations, including macros for the inference of tasks and the generation of diagrams by creation/format and specification/content.

Conceived with a didactic approach, this computational framework is implemented as open source and multi-platform software, using the Python programming language. The development stages of the VPE-GM framework consisted of the construction and implementation of the memory and process editors, simulator, and related graphic interfaces, including the external representation modules. Using the visual process editor, the files can be created, saved, deleted, built, and changed (Figure 4). These functions are implemented by the concept of *envelopes*, files created in Python for the internal representation of processes and characterized by the grouping of graphic objects on the interface of the process editor.

Memory configurations are built in the *memory editor*. The modelling of the global shared memory by a matrix structure is represented, generally, by a discrete subset of the Euclidian Geometric Space. The memory value indexation is explicit in the graphic representation, which makes the application and understanding of parallelism and non-determinism easier. Besides, it allows the visualization of the simulation, see the interfaces in Figure 4.

In the simulator, the computations can be performed in different options in order to present the results of partial computations and at the same time the validation of the program. Other available options supported are based on methodology of step by step and the pre-selection of the subset of check points. Then the edited program and memory configuration are both previously built by their interfaces and saved as an XML file.

Moreover, the visual programming environment provides two modules which carry out the integration of each editor with the simulator. These modules, named external representation modules, enable a program or a memory as an XML file. Such files preserve all the features, either of a program or of a memory, described in a compatible way.

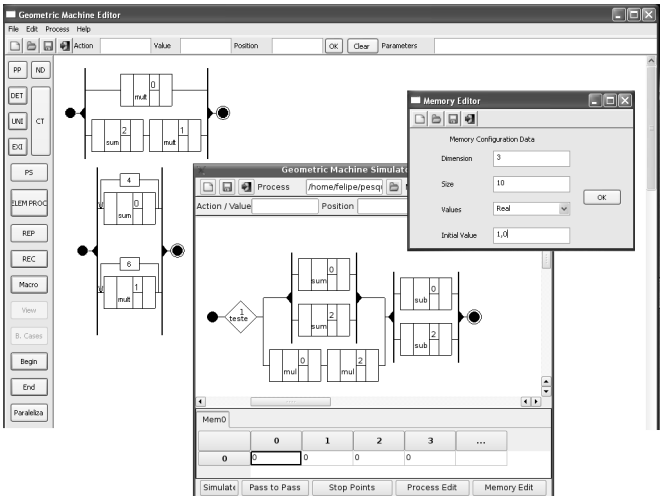

Figure 4: Snapshots of the interfaces in the VPE-GM.

In addition, the simulator is able to interpret them and check the semantic specifications. Then it defines the necessary constructions to achieve the simulation or to export such files to the execution environment. In Figure 5, the Python numerical library was used to make an instance of such construction. In addition, the XML file presents the data related to the function parameters and some fields (color and dimension) which are used for the graphic representation. Later, the user should configure the memory which will be used in the simulation in a way that it is compatible with the program created on the process editor interface. In order to check this compatibility, it is necessary that the memory shows all the positions referred to graphic construction. If a selected memory was built, the related XML file allows the visualization of the obtained result as well as the data presentation concerning the memory position and the stored values.

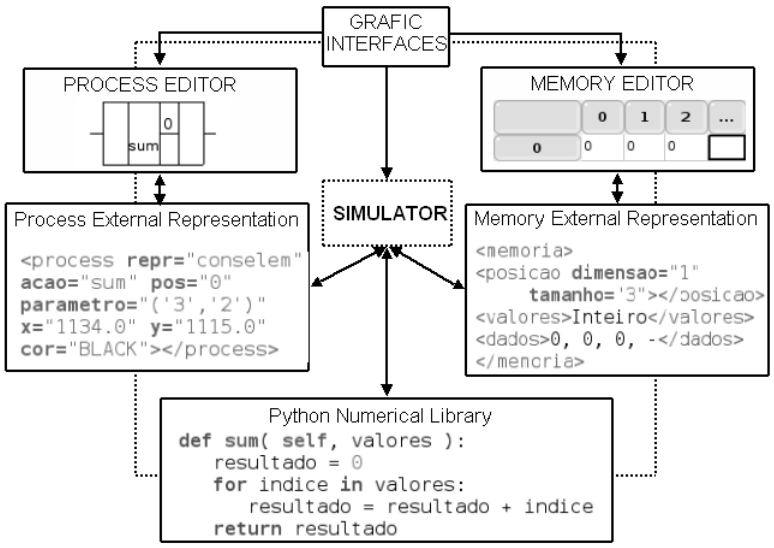

Figure 5: Operation model in the VPE-GM.

4 Parallel abstractions in the GM model

In this paper, the study of potentialities and exploration of the parallelism, according to the categorization dependent on the levels of abstraction proposed by [13], are focused on two main objectives: (*i*) systematize the potentialities

for the exploration of parallelism in the GM model; (*ii*) consolidate the definitions for the process constructors of the GM model.

Conceived as a low-level model, it makes all the issues of parallel programming explicit. The GM abstraction model consists of a set of processes labelled by positions of a geometric space, capable of executing independent program in a synchronized way, connected to a shared memory. All processes can access any location in the unit of time but they can not access the same location on the same step of computation. The GM model requires very detailed description, giving the code and memory position for each process as well as ensuring the treatment for memory conflict by a procedure called position-function. The decomposition, mapping and synchronization should be specified by the software developer.

The address space is characterized as a multidimensional matrix structure, topologically compatible with the geometric space. The symmetric processes synchronization enables a better use and flexibility of the computational resources. In addition, they allow reconfiguration of the system if there is a failure in the process performance.

Characterized as a static RAM memory and parallel memory structure shared by the processors, the model provides synchronous execution of concurrent data and exclusive write. The uniform memory access of the global memory may be applied to multiple instruction and multiple data.

Thus, GM machine is conceived as a concrete model, which may provide predictable performance but makes it hard to construct and design software. Recognizing the importance of abstraction, this work introduces the virtual approach of the GM model, denoted by VirD-GM, to maintain the portability and the maximum expressiveness to be efficiently implemented.

The challenge is to use theoretical analysis in order to achieve practical experimentations. For that, the execution management module in the Vird GM decreases the difficulties in the task of parallel programming, as described in the following section.

5 Distributed approach to GM model

The evolution and consolidation of D-GM model constitutes an opportunity to introduce an execution environment over which distributed and parallel computations can be improved in their sequential performance, as presented in Figure6. The focus of this research, VirD-GM, aims to construct a virtual distributed machine as a support for computations in the GM model. Based on the specification of the GM model, the use of the EXEHDA middleware is considered to implement this virtual distributed machine. More specifically, the XML files described in the last section, are generated and exported by the VPE-MG to the VirD-GM.

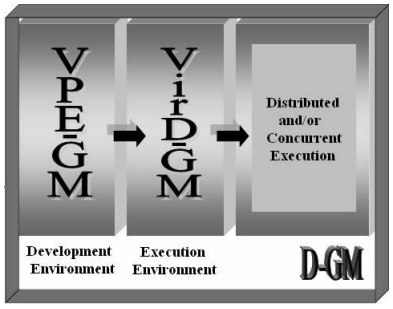

Figure 6: Functional Vision of Distributed approach to GM model

The results in [18] indicated that EXEHDA could support the demands of distributed/parallel execution, memory access, and communication introduced by the VirD-GM model. In that sense, EXEHDA is a pervasive computing middleware [11, 15, 2, 3] based on contextualized services which assist two perspectives [17]: *(i)* management and creation of the distributed environment, by providing services to control the physical medium (resources) where the processing will take place; and *(ii)* support for execution of application, by providing the services and abstractions required for implementing large-scale distributed applications. Middleware services are conceptually organized in a set of subsystems: data and code pervasive access, uncoupled spatial and temporal communication, large-scale distribution along with context recognition and adaptation. The integration of these subsystems is shown in Figure 7.

A key abstraction in EXEHDA for supporting VirD-GM computations is the *OX* (Object eXehda). It consists of an object, instantiated through the *Executor Service*, which can be bound to arbitrary (text or binary) execution attributes [16, 18]. Moreover, these attributes are made available in a distributive way in the system by the *OXManager*

service. On the other hand, the subset of services of the subsystem *Data and Code Pervasive Access* in EXEHDA is capable to manage the GM model memory requests related to spatially distribution, conflict of memory access, synchronization, validation of memory positions and stored values.

The EXEHDA components applyed in the VirD-GM are not only the basic services for supporting distribution, communication and synchronization, but also the high level issues concerned with the login, the scheduling and the security services.

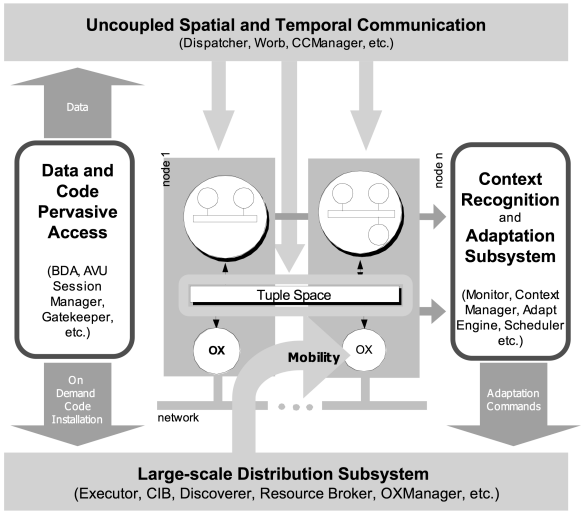

Figure 7: Integration of EXEHDA subsystems

6 Conclusion

This paper presents the evolution of GM model research from the domain-theoretic structure of concurrent processes to the target objective to promote their real distributed computations. Following this methodology, over the visual approach of the GM model is defined the development environment.

The VPE-GM model is induced by the ordered structured and semantic foundation of the GM model. This work integrates specification and implementation of parallel programs, considering the basic concepts of the parallelism proposed in [14].

The VPE-GM is easy to program and understand, and its methodology follows the inductive construction of ordered structure of the GM model. It supports decomposition of parallel programs and process synchronization. A possible application is the analysis of the parallel and distributed algorithms of Interval Mathematics based on the Interval Geometric Machine model, a distributed extension of the GM model [7, 8, 9].

This paper also introduces an ongoing work, named VirD-GM, the virtual approach for the GM model over which is possible to support real distribution and concurrence. It provides support for implementation of parallel programs in a higher level of abstraction, using the EXEHDA middleware.

References

[1] BARDOHL, R., AND ERMEL, C. Visual specification and parsing of a statechart variant using genged. In *Proc. Symposium on Visual Languages and Formal Methods* (2001), pp. 5–7.

[2] DAVIES, N., FRIDAY, A., AND STORZ, O. Exploring the grid's potential for ubiquitous computing. *IEEE Pervasive Computing 3*, 2 (2004). New York.

[3] DE ROURE, D., JENNINGS, N., AND SHADBOLT, N. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley & Sons, New York, 2003, ch. The Evolution of the Grid.

[4] GIRARD, J. Y. Linear logic. In *Theoretical Computer Science*. 1987, pp. 1–102.

[5] PRESTES, D., REISER, R., COSTA, A., AND CARDOSO, M. Extending the geometric machine model to a visual programming environment. In *CLEI2005 XXXI Conferência Latinoamericana de Informática* (Cali, 2005), Universidad Javeriana, pp. 1–10. meio digital.

[6] REISER, R., COSTA, A., AND DIMURO, G. First steps in the construction of the geometric machine model. In *TEMA - Tendências em Matemática Aplicada e Computacional*, vol. 3. 2002, pp. 183–192.

[7] REISER, R., COSTA, A., AND DIMURO, G. A programming language for the interval geometric machine model. In *Eletronic Notes in Theoretical Computer Science*, vol. 84. 2003, pp. 1–12.

[8] REISER, R., COSTA, A., AND DIMURO, G. The interval geometric machine. In *Numerical Algorithms*, vol. 37. Kluwer, Dordrecht, 2004, pp. 357–366.

[9] REISER, R., COSTA, A., AND DIMURO, G. The distributed interval geometric machine model. In *Lecture Notes in Computer Science* (2005), no. 3732, Springer, pp. 179–188.

[10] REISER, R., COSTA, A., DIMURO, G., AND CARDOSO, M. Specifying the geometric machine visual language. In *IEEE Symposium on Visual Languages and Formal Methods* (2003), pp. 1–3.

[11] SAHA, D., AND MUKHERJEE, A. Pervasive computing: a paradigm for the 21st century. *IEEE Computer 36*, 3 (March 2003), 25–31.

[12] SCOTT, D. The lattice of flow diagrams. In *Lecture Notes in Mathematics*. Springer Verlag, 1971, pp. 311–372.

[13] SKILLICORN, D. *Fundations of Parallel Programming*. Cambridge: University Press, New York, January 1994.

[14] SKILLICORN, D. B., AND TALIA, D. Models and languages for parallel computation. *ACM Computing Surveys 30*, 2 (june 1998), 123–169.

[15] WESNER, S., DIMITRAKOS, T., AND JEFREY, K. Akogrimo - the grid goes mobile. *ERCIM News, Special Themes: Grids - the next generation*, 59 (Octobe 2004). New York.

[16] YAMIN, A. C., AUGUSTIN, I., BARBOSA, J., SILVA, L., REAL, R., CAVALHEIRO, L., AND GEYER, C. Towards merging context-aware, mobile and grid computing. *Journal of High Performance Computing Applications 17*, 2 (2003), 191–203.

[17] YAMIN, A. C., AUGUSTIN, I., BARBOSA, J., SILVA, L., REAL, R., AND GEYER, C. A framework for exploiting adaptation in high heterogeneous distributed processing. In *14th Symposium Computer Architecture and High Performance Computing* (Vitoria).

[18] YAMIN, A. C., AUGUSTIN, I., BARBOSA, J., SILVA, L., REAL, R., SCHAFFER, A., AND GEYER, C. Exehda: adaptive middleware for building a pervasive grid environment. In *Frontiers in Artificial Intelligence and Applications - Self-Organization and Autonomic Informatics*, H. Czap, R. Unland, C. Branki, and H. Tianfield, Eds., vol. 135. IOS Press, 2005, pp. 203–219.