Defining and Executing Bigraphical Model in Maude

Manel Amel Djenouhat, Taha Abdelmotaleb Cherfia, Faiza Belala

DEFINING AND EXECUTING BIGRAPHICAL MODEL IN MAUDE

Manel DJENOUHAT [1], Taha Abdelmoutaleb CHERFIA [2] and Faïza BELALA [3]
Computer Science Department, University Mentouri of Constantine,
Algeria.
[1] djenouhat_manel@msn.com, [2] taha.cherfia@gmail.com , [3] belalafaiza@hotmail.com

ABSTRACT

Software architectures can help in specification, formal analysis and manipulation of complex and adaptive systems. Many languages and formalisms have been proposed for these tasks, especially those based on graph model. Bigraphical Reactive Systems (BRS) are an emerging graphical framework for specifying ubiquitous dynamic architectural systems.

In this work, we propose using the K semantic framework, a provably sound and complete modeling methodology, to integrate BRS into rewriting logic. This approach offers a high level specification of bigraphical systems without any encoding or translation process. Moreover, we are also able to formally reason on it.

Finally, we illustrate the use of the K-Maude tool through a simple example of a packet transport.

General Terms
Software engineering, Software architecture systems, Formal methods.

Keywords
Software Architecture, Bigraph, Rewriting Logic, K-Maude

I. INTRODUCTION

Complex and adaptive systems challenge software engineering to deal with issues like scalability, dynamicity and openness at the right level of abstraction. In this context, software architectures can help in the specification, formalization and manipulation of such systems by restricting and disciplining the admissible shapes and patterns to be considered. Many languages and formalisms have been proposed for these tasks, such as graph-based models which represent a strong and rigorous approach to software architecture design. Indeed, graphs are a very useful means to describe complex structures and systems and to model concepts and ideas in a direct and intuitive way. In particular, they provide a simple and powerful approach to a variety of problems that are typical to software architecture. Besides, if graphs define the structure of such models, graph transformation can be exploited to specify both how they should be built and how they can evolve.

Among these graphical mathematical formalisms, Bigraphical Reactive System (BRS) is an emerging framework, proposed by Milner and others [1], [2] as a unifying theory of process models for distributed, concurrent and ubiquitous computing. A bigraphical reactive system consists of a category of bigraphs (usually generated over a given signature of controls) and a set of reaction rules. Bigraphs can be seen as representations of the possible system configurations, and the reaction rules specify how these configurations may evolve (i.e. the relations between bigraphs). Bigraphs, in addition to their wide use in modeling various systems (λ-calculus systems [3], CCS [4], Petri nets [5] and ambient calculus systems [6]) have been recently shown as an efficient framework for specifying architectural systems [14], [15], [16].

However, formal specifications based on Bigraphs would be a mere luxury, so that their execution and analysis must be accomplished in isolation. Therefore, they can be proved to be correct or will do what they are supposed to do, but involving other approaches with additional formalisms and formal tools. The goal in this regard is to express as faithfully as possible a bigraphical system in rewriting logic avoiding thus, any encoding or translation process. Hence, this will allow the combination of the attractive advantages of the two models. Bringing different models under a common semantic framework makes easier to understand what different models have in common and how they differ, to find deep connections between them and to reason across their different formalisms. Furthermore, rewriting logic (RWL) has shown to be a good framework in which other logics and a very wide range of concurrency models and programming languages can be represented, such as Petri nets [5], E-LOTOS [10], CCS [4], π-calculus [11] and so on. Computationally, it supports also the execution of specified models, their prototyping and analysis thanks to many operational environments developed around its theoretical concepts, Maude [7], Elan [8], Cafe-OBJ [9]. Many different system styles, and models of concurrent computation and many different languages can be naturally expressed without any distorting encodings.

The main objective of our contribution is to achieve, in a rigorous way, the integration and interoperation of BRS in K-Maude [12] which is an effective computational framework for systems definition and formal analysis. This allows to define executable and analyzable formal specifications of ubiquitous dynamic architectural systems.

The remainder of the paper is organized as follows. We start by introducing the Bigraphical Reaction Systems (or BRS) in section two. In section three, we recall fundamental elements of K language and their integration in Maude system. In section four we present our approach. The proposed solution of specifying a BRS with K annotations and the possibility of analyzing the result on K-Maude is given. Then, we demonstrate our contribution through the design of an illustrative example. Finally, we conclude the paper in section five.

II. SPECIFYING WITH BRS

The Bigraphical Reaction System or BRS in short, due to Milner and co-workers [1] is a graphical model in which both locality and connectivity of distributed systems are prominent.

Informally, a bigraph consists of two independent structures (see figure 2), a place graph (topograph) expressing the physical location of nodes (v_i) and a link graph (monograph) representing the interconnection (e_i) between these nodes.

Moreover, bigraphs are equipped with a set of reaction rules to form BRS these rules may be applied to rewrite bigraphs. Thus, reaction rules define the dynamics of bigraphs where two transformations are possible, nesting and linkage. BRS allow the specification of both structural and behavioral aspects of architectural systems.

II.1 Structure aspect

A bigraph is a structure that enables the description of both the location of the architectural system entities and their interaction. Within BRS, these entities are represented by nodes and the interaction between them is represented by edges as shown in figure 1 below:

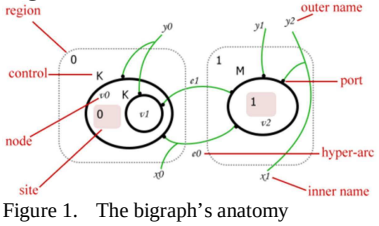

Figure 1. The bigraph's anatomy

On the basis of a common set of nodes, a bigraph is formed of two independent structures (see figure 2): the places graph, having the structure of a forest that shows the spatial distribution of the application, the links graph is a *hypergraph* establishing the model of connectivity between different nodes.

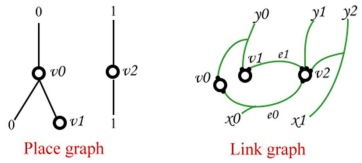

Figure 2. Places graph and link graph

While an arc in the places graph shows the relationship of spaces between the nested elements of the application, an arc in the graph links establishes a connection between the ports of these elements. The two structures are orthogonal, so links between nodes can cross locality boundaries. Each tree in the places graph represents a region (0 or 1 in figure2) of space that can contain sites, corresponding to the leaves of the tree, and where other bigraphs can be hosted.

A bigraph can interact with the environment through its interfaces (x_i, y_i). These interfaces are known as inner face and outer face where sites and inner names make up the bigraph's inner face, while roots and outer names model the outer face.

The inner faces of a bigraph indicate the holes into which other bigraphs can be inserted. On the other hand, the outer faces allow a bigraph to be combined with other ones.

Figure 3 shows a bigraph which classifies nodes as computers (*PCi*), packets (*P*), networks (*NET*) and represents a system state which may change because of the transport of packets.

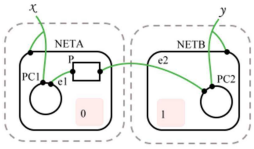

Figure 3. Bigraph structure aspect

II.2 Dynamic aspect

Within BRS, the dynamic behavior of the architectural system is modeled based on reaction rules which express how the system can reconfigure itself.

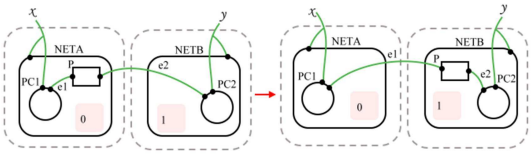

Figure 4. Bigraph dynamic aspect

Each reaction rule consists of a redex which may be transformed to a reactum to rewrite the bigraph.

For example in figure 4, the reaction rule represents a one-way packet transport where *P* transports from the source computer (*PC1*) in the network (*NETA*) through the destination computer (*PC2*) in a different network (*NETB*). The hyper-arcs *e1* and *e2* model the path of the packet traffic. Thus, this reaction rule is purely a nesting transformation.

Some works in literature have shown that bigraphs form an unifying framework for concurrency and mobile models, such as CCS, π-calculus, ubiquitous systems or Petri-nets [13]. Recently, authors respectively in [14] and [15] show that BRS constitute a suitable mathematical framework for modeling both architectural application and their possible execution platform. BRS gives a high-level modeling of architectural systems in terms of graphical representation which contains both the information about the architectural entities hierarchies and the interaction between them.

Furthermore, authors illustrate in [14], [16], how BRS can be used to formalize the architectural deployment and dynamic reconfiguration of AADL systems.

III. K-MAUDE: A REWRITING BASED TOOL FOR COMPUTATIONS

K is an executable semantic framework in which programming languages, calculi, as well as type systems or formal analysis tools can be defined, making use of configurations, computations and rules.

Configurations organize the system/program state in units called cells, which are labeled and can be nested. *Computations* carry ``computational meaning" as special nested list structures sequentializing computational tasks, such as fragments of program; in particular, computations extend the original language or calculus syntax.

K rules generalize conventional rewrite rules by making explicit which parts of the term they read, write, or do not care about.

This distinction makes K a suitable framework for defining truly concurrent languages or calculi, even in the presence of sharing and control-intensive language features such as abrupt termination, exceptions, or call/cc (call with current continuation function).

K-Maude is the tool implementing K on the top of Maude. It fully extends Maude, while adding specific K constructs to facilitate design of programming and domain specific languages.

III.1 Maude Overview

Maude **[7]** is a high-level language and a high-performance system supporting executable specification and declarative programming in rewriting logic. It is based on rewriting logic where the systems, from simple to more complex, are specified easily by the use of the functional, concurrent or object oriented theories. Thus, Maude regroups three types of modules mainly: *functional modules* to define the static aspects of a system, *system modules* to specify the dynamic aspect of the system using rewriting rules, while *object oriented modules* allow the modeling of the objects oriented systems.

Maude supports in a systematic and efficient way logical reflection. This makes it remarkably extensible and powerful, it supports an extensible algebra of module composition operations, allows many advanced meta programming and meta language applications.

III.2 K Basic Concepts

K language definitions are represented as *K-modules*, fully inspired from Maude modules. Each K module imbricates two sub-modules: *k-syntax-module* and *k-module*. This separation reduces ambiguities in the parser and therefore might be able to parse a large variety of programs.

K-syntax-module is introduced by kmod <name module>-SYNTAX …endkm, it represents the grammar of the specified language (see example of figure 5).

```
kmod EXP−SYNTAX
//@ Arithmetics Syntax
syntax Exp ::= #Int
| Exp "+" Exp [strict] //addition
| Exp "*" Exp [strict] //multipl
| Exp "/" Exp [strict] //division
| Exp "?" Exp ":" Exp [strict(1)]
| Exp ";" Exp [seqstrict]
 //@ Input / Output Syntax
 syntax Exp ::= "read"
| "print" Exp [strict]
//@ Concurrency features
syntax Exp ::= "spawn" Exp
| "rendezvous" Exp [strict]
endkm
```

Figure 5. K-syntax module of a calculator language with I/O

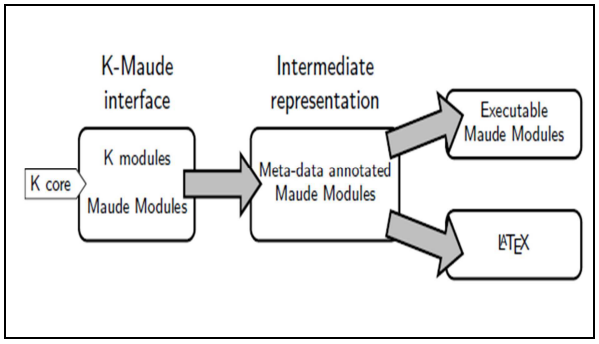

Figure 6. K-Maude Architecture

k-module is introduced by kmod<name module> …endkm (figure 7) and represents the semantic of the grammar quoted previously. It contains three principal parts: *Evaluation strategies (Strictness)* which represent the link between syntax and semantic and the order in which the arguments of a construction must be estimated, *Configurations* representing the state of a system during its execution, *Rules* describing how a current configuration evolves.

```
kmod EXP
imports EXP−SYNTAX
syntax KResult ::= #Int
configuration
<k color="green" multiplicity="*"> $PGM:K </k>
<streams>
<in color="magenta" stream="stdin"> .List </in>
<out color="Fuchsia" stream="stdout"> .List </out>
</streams>
//@ Arithmetics Semantics
rule I1:#Int + I2:#Int => I1 +Int I2
rule I1:#Int * I2:#Int => I1 _Int I2
rule I1:#Int / I2:#Int => I1 /Int I2
when I2 =/=Bool 0
rule 0 ? _ : E:Exp => E
rule I:#Int ? E:Exp : _ => E when I =/=Bool 0
rule _:#Int ; I2:#Int => I2
//@ Input / Output Semantics
rule <k> read => I:#Int ···</k>
<in> ListItem(I) => . ···</in>
rule <k> print I:#Int => I ···</k>
<out>··· . => ListItem(I) </out>
//@ Concurrency Semantics
rule <k> spawn E => 0 ···</k>
(. => <k> E </k>)
rule <k> rendezvous I => 0 ···</k>
<k> rendezvous I => 0 ···</k>
endkm
```

Figure 7. K- module of a calculator language with I/O

Examples in figures 5 and 7 show the global k module of the simple calculator language with input and output (the whole explanation of the example is included in [17]).

III.3 Application of K-Maude tool

K-Maude is an integrated toolkit on the top of Maude system. Figure 6 shows its architecture [12]. The gray arrows represent translators implemented as part of the toolkit. The *K core* contains the ingredients of the K technique that are handy in most language definitions, such as ones for defining computations, configurations, etc. The K-Maude *interface* is what the user typically sees: besides usual Maude modules (K-Maude fully extends Maude), one can also include K-Maude files (with extension .kmaude or .k) containing modules using the K specialized notation.

A first component of K-Maude translates K modules to Maude modules. The resulting Maude modules encode K-specific features as *meta-data* attributes and serve as an intermediate representation of K-Maude definitions. This intermediate representation can be further translated to different back-ends. We provide two such translators, one to *executable/analyzable* Maude module results which can serve as interpreters for the defined languages or as a basis for formal analysis and one to LATEX for documentation purposes.

Indeed, we believe that K can be used by ordinary language designers as a formal notation for rigorously specifying the semantics and grammars (syntax) of their languages.In the same way context-free, we present a novel approach based on the use of K techniques, we propose an inductive proving method which tends to provide a full specification for bigraphical reactive systems.

Our modeling process is as follows: Starting from writing K specifications of the BRS system using the proposed model we obtain a comprehensive K model of this one. Then, we may transform it into Maude theories, the intermediate Maude representation resulting can be analyzed and executed using the Rewriting logic framework, this gives us the possibility to determine whether the bigraphical model of system is correct and satisfies different properties in all its possible executions.

IV. OUR IMPLEMENTATION APPROACH

In this section we present our adopted method to represent Bigraphical architecture systems by using K-Maude.First, we give K definitions of a bigraph , this aim at defining a specification language using one's favorite semantic style. Once the K model obtained, we may execute, explore or model check any bigraphical system example thanks to Maude system.

IV.1 Our contributions

Knowing that K is specially introduced to describe programming languages, as result, we define an appropriate grammar for bigraphs and adapted it to K framework. We are also inspired by the BigMC model checker [18]. The full grammar for bigraph terms is given in figure 8, where a bigraph is represented as a list of expressions.

```
module BIGRAPH-SYNTAX
imports BUILTIN-SYNTAX-HOOKS
syntax Bigraph ::= List{Exp,","}
syntax Exp ::=    Exp "." Exp
            | Exp "|" Exp
            | Exp "||" Exp [strict (1)]
            | "(" Exp ")"       [bracket]
            | "[" Exp "]"       [bracket]
            |"Nil"
            | "passive" Exp ":" Int
            | "active" Exp ":" Int
            | "name" Id
syntax Redex ::= Bigraph
syntax Reactum ::= Bigraph
syntax Reactrule ::= Redex "->" Reactum
endmodule
```

Figure 8. Full syntax module of a bigraph in K

We used the term Expression to specify globally all terms used for describing a bigraphical system such as: nodes, links and also controls which are pre-defined by the declaration of the bigraph *signature* using **active** and **passive** commands defining the arity of a given control as well as it's a passive or active state. On the other hand, we have also give syntax of reaction rules by introducing redex and reactum bigraphs (the left and the right parts of the rule) then, we complete our defined syntax by introducing its semantic in the main module BIGRAPH.k. Figure 9 gives the complete semantic of our Bigraphical specification, the configuration provides the new state after each execution, as mentioning in KResult, each new bigraph resulting a new bigraph from the application of K rewriting rules allowing us to guide the execution.

```
module BIGRAPH imports BIGRAPH-SYNTAX
configuration
                < T color ="yellow">
                <k color="green">.K </k>
                <state color ="red">.Bigraph </state>
                </T>
syntax KResult ::= Bigraph
rule  [terms] : T1:Exp . T2:Exp => T1 . T2
rule  [juxtaposition] : T1:Exp | T2:Exp  => T1 | T2
rule [sameregion] : T1:Exp || T2: Exp => T1 || T2
rule [reactrules]:  T1:Exp.(T2:Exp|T3:Exp)|| T4:Exp .(T5:Exp) =>
T1:Exp.(T2:Exp)|| T4:Exp .(T5:Exp|T3:Exp)
endmodule
```

Figure 9. Global K module BIGRAPH for Bigraphs systems

After defining the Main specification module we may compile it with the compiler Kompile(see figure 10), thus, we obtain its equivalent Maude module. The resulting Maude module encodes K-specific features of bigraphs as meta-data attributes.

```
/home/tools/k-tool/core/kompile BIGRAPH.k  -l  BIGRAPH

Compiled version written in BIGRAPH-compiled.maude.
```

Figure 10. Compiling BIGRAPH.k module

Once the program (the bigraphical model) is compiled, we can run it easily on Maude using the Krun tool. We may also use Latex transformer to get the PDF version of the program.

IV.2 Example

We illustrate our approach by introducing the previous bigraph example of transferring packets between computers nested in two different networks.

```
module BIGRAPH-PROGRAMS imports BIGRAPH
syntax Id ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|$0|$1|e1|e2

//nodes and links

syntax Exp::= active NETA:1
            |active NETB :1
            |active PC1:2
            |active PC2:2
            |active P :2
            |name x
            |name y
            |name e1
            |name e2

syntax Bigraph ::=BigNet

// packet-Transport Model

BigNet::= NETA[x].(PC1[x,e1]|P[e1,e2])|NETB[y].PC2[y,e2]

//reaction rule

Redex  ::=  NETA  [x]  .($0|PC1[x,e1]|P[e1,e2])  ||
NETB[y].($1|PC2[y,e2])

Reactum::=NETA[x].($0|PC1[x,e1])||NETB[y].($1|PC2[y,e2]|P[e1
,e2])

endmodule
```

Figure 11. K representation of Packet transport model

According to the proposed model we obtain the Bigraph BigNet representing the two networks and also the applied reaction rule. Running this example on K-Maude, we get the following displayed on the console:

```
rewrite in BIGRAPH : [['BigNet]] .
rewrites: 799  in 0ms cpu (1ms real) (~ rewrites/second)
result Bigraph:
<T>
  <k>
    .
  </k>
  <state>
BigNet |->
Redex::=
NETA[x].(PC1[x,e1].Nil|  P[e1,e2].Nil)  |NETB[y ].(PC2[y,e2].Nil
Reactum::=
NETA[x].(PC1[x,e1].Nil)|NETB[y ].(PC2[y,e2].Nil|P[e1,e2].Nil)
  </state>
</T>
```

Figure 12. The K running of the Packet transport example

This shows that *PC2* in the network *NETB* receives the transferred packet *P* from *PC1* in the network *NETA.*

V. CONCLUSION

Recently, the complexity and the size of software systems have increased substantially. Thus, software architecture is the needed solution to deal with these issues. In this context, many modeling formalisms and languages have been proposed to represent software architectures such as graph-based formalisms. In the same context, BRS represents a high-level modeling graph-based formalism in terms of graphical representation whereas it lacks at present a complete formal execution framework.

Hence, this paper presents a new modeling methodology based on K-Maude to integrate BRS model into rewriting logic. Indeed, the main issue with the proposed methodology is to define a semantic execution environment for BRS models on the basis of Maude language. Looking ahead, we intend to exploit this new semantic model of bigraphs for executing architecture and verifying the correctness of some properties inherent to dynamicity and reconfiguration of these systems. Also, we shall present more examples and comparisons to other related works.

REFERENCES

[1] R. Milner. *"Bigraphical Reactive Systems"*. In Larsen and Nielsen (eds.), Proc. 12th CONCURS. Lecture Notes in Computer Science 2154, pp. 16–35. Springer, 2001.

[2] R. Milner. *"Pure bigraphs: Structure and dynamics"*. Information and Computation 204(1):60–122, 2006.

[3] G. Michaelson, *"An Introduction to Functional Programming through Lambda Calculus"*, Addison-Wesley, Wokingham, 1988.

[4] R. Milner *"Calculus of Communicating Systems"*,1989

[5] J. J. Leifer, R. Milner. *"Transition systems, link graphs and Petri nets"*. Mathematical

[6] Cardelli, L., G. Ghelli and A. D. Gordon, *"Types for the ambient calculus"*, Information and Computation (2002), to appear. Special issue on TCS'2000.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. *"All About Maude"*, A High-Performance Logical Framework, volume 4350 of Lecture Notes in Computer Science. Springer, 2007.

[8] P. Borovanský, C. Kirchner, H. Kirchner, P. E. Moreau, and C. Ringeissen. *"An overview of ELAN"*. Electr. Notes Theor. Comput. Sci., 15, 1998.

[9] R. Diaconescu and K. Futatsugi. *"Logical foundations of CafeOBJ"*. TCS, 285(2) :289-318, 2002.

[10] J. Quemada, editor. "*Working Draft on Enhancements to LOTOS*". Draft International Standard, ISO/IEC JTC1/SC21/WG7 Project 1.21.20.2.3, January 1997.

[11] M. Bundgaard, V. Sassone. *"Typed polyadic pi-calculus in bigraphs"*. In Bossi and Maher (eds.), Proc. PPDP. Pp. 1–12. ACM, 2006.

[12] T. Serbanuta, G. Rosu *"K-Maude: A Rewriting Based Tool for Semantics of Programming Languags"*, *WRLA'10*, LNCS 6381, pp 104-122. 2010.

[13] R. Milner, *"Bigraphs and Their Algebra"*. Electr. Notes Theor. Comput. Sci. 209: 5-19 (2008).

[14] N. Benlahrache, F. Belala, K. Barkaoui, "Description formelle du déploiement d'architectures AADL basée sur les systèmes réactifs bigraphiques (BRS)", CALL'2011, *5ème Conférence Francophone sur les Architectures Logicielles*, Lille, France, 2011, pp. 65-75.

[15] Z. Chang, X. Mao, Z. Qi *"An Approach based on Bigraphical Reactive Systems to Check Architectural Instance Conforming to its Style*", *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE'07)*, pages 57-66. IEEE Computer Society

[16] T. Cherfia, F. Belala, N. Benlahrache *"Modeling of Architectural Reconfiguration Case Study: Automated Teller Machine"*, *IWAISE'12.*

[17] T. Serbanuta, A. Arusoaie, D. Lazar, C. Ellison, D. Lucanu, G. Rosu. *"The K Primer (version 2.5)"*, Technical Report, 2012.