A Tile Logic Based Model for a Collaborative Session Application

C. Bouanaka , F. Belala, and A. Choutri

LIRE Laboratory, Department of Computer Science, Mentouri University. Constantine, Algeria.

Tel./Fax: 213 31 81 88 88

{Bouanaka2006, ChoutriAicha}@yahoo.fr; Belalafaiza@hotmail.com

Abstract — **In a previous work, we have defined an architectural model for software architectures description based on Tile logic. We have also defined a dynamic connection between components, which is invoked only if an interaction is needed. Our aim in the present work is to show the expressive power of the proposed model to specify realistic applications, such as collaborative session application.**

Index Terms — **Tile Logic, Architecture Description Languages, Synchronization, Collaborative Applications.**

I. INTRODUCTION

Nowadays, software engineering community is developing techniques centered on systems architectures description in order to improve the understanding and the conception of complex systems, to encourage their evolution and their reuse, and to proceed to various analysis. These techniques are materialized by specific languages, ADLs (Architecture Description Language) [1], which allow a software designer to focus on high-level aspects of an application by disregarding details of components that contribute in the architecture. It is precisely this abstraction that makes ADLs suitable for verification using model checking techniques.

A great number of ADLs have been proposed in the literature. However, most of them: Wright [2], Rapide [3], Darwin [4], etc., focus on the software architecture description where component semantics is in part expressed by its interface, and system behaviour is not completely defined. Therefore, software architecture concepts need to be associated to formal theories, clarifying these concepts or providing rules to determine whether a given architecture is well-formed.

In our model [5], system software architecture, designed to facilitate designers job, is systematically transformed to a formal theory specification, which can be prototyped or model checked. This facilitates the integration of formal specifications in the traditional life-cycle of an application development. We present an interesting combination of Tile logic [6], an extension of rewriting logic [7], and software architectures to define *Tile logic based model* of an ADL inherent concepts [5]. It considers the system software architecture as a set of black boxes interconnected via an interconnection topology. It also allows defining alternative transparent boxes where internal behaviours can be formally specified.

The remainder of the paper is organized as follows. Section 2 begins by motivating Tile logic choice as a semantic framework for our model. Then, basic semantic aspects of Tile Logic are presented. Finally, the collaborative session case study formalisation and the main ideas introduced on software architectures description are specified. Discussion and conclusions round out the paper.

II. A FORMAL MODEL FOR COLLABORATIVE SESSIONS

Distributed collaborative applications are characterized by supporting groups' collaborative activities. This kind of applications is branded by physically distributed user groups, who cooperate by interactions and are gathered in work sessions [8]. The effective collaboration result is a production of simultaneous and concurrent actions, carried out during the definition and the execution of the session. Thus, interaction plays the prominent role in collaborative sessions and requires being coordinated (synchronized) to avoid inconsistencies. Consequently, adopting an approach that follows a component-based software development and that exploit a clean conceptual separation between computation and coordination is recommended. Tile model is the adequate framework since it is based on a configuration notion, that includes input and output interfaces where actions can be observed and that can be used to compose configurations and also to coordinate their local behaviours [8].

A. Tile logic

Tile logic [6] is an extension of rewriting logic (in the unconditional case) taking into account rewriting

with side effects and rewriting synchronization. The main idea is to impose dynamic restraints on terms to which a rule may be applied by decorating rewrite rules with observations ensuring synchronizations and describing interactions. The resulting rewrite rule is called a tile.

A tile $\alpha : s \xrightarrow[b]{a} t$, and represented graphically in Fig.2, is a rewrite rule stating that the initial configuration *s* can evolve to the final configuration *t* via α, producing the effect *b*; but the step is allowed only if the arguments of *s* can contribute by producing *a*, which acts as the trigger of α. Triggers and effects are called observations.

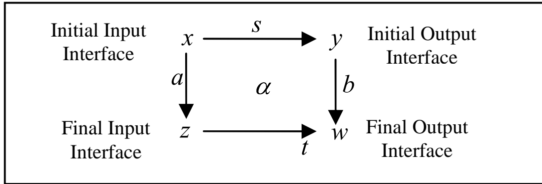

Fig. 1. Graphical Representation of a Tile

Definition [6]: A *tile system is a 4-tuple* ***R*** *= (H, V, N, R) where H, V are monoïdal categories with the same set of objects* $O_H = O_V$, *N being a set of rule names and R: N→ H x V x V x H a function where for each α in N, if R(α) = (s,a,b,t), then s:x→y, a: x→z, b:y→ w and t: z→w, for suitable objects x, y, z and w, x and z are the input interfaces. While, y and w are the output interfaces.*

Since Tile logic exploits three dimensional views, tiles can be composed horizontally, in parallel, or vertically to generate larger steps. Horizontal composition $\alpha;\beta$ coordinates the evolution of the initial configuration of α with that of β yielding the synchronisation of the two rewrites [9]. Horizontal composition is possible only if the initial configuration of α and β interact cooperatively: the effect of α must provide the trigger for β. The parallel composition builds concurrent steps. Vertical composition is the sequential composition of computations.

B. COLLABORATIVE SESSION FORMALISATION

In the proposed model, our collaborative session example is composed of one president and several instances of participant component. President component interacts with each participant component via a pair of (input/output) ports. A connection between (input/output) port of the president and (output/input) port of a participant is established dynamically if needed in the sense that individual components definition of a the collaborative session is completely independent from their interconnection topology. Thus, components are viewed as floating elements since static interconnection topology definition, like in most existing ADLs, is completely absent.

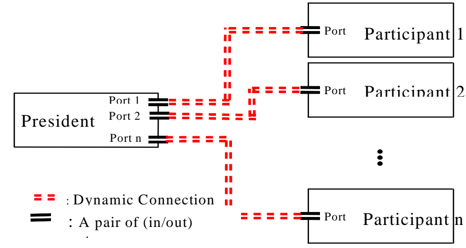

Fig. 2. Software Architecture of a collaborative Session

C. Components internal structure

Each component is defined as a set of external ports, to ensure interactions with the environment, and an internal behaviour operating on its basic structure.
The intended behaviour of the collaborative session is as follows: To open a session, the president begins by announcing it. He prepares an invite message and sends it to each participant. The session is opened by the president when at least a positive response (accept to participate) is received. Participants are then informed by an open message. Managing session consists of realizing the collaborative task with contribution of all session members. The president closes the session, by sending a close message to participants, when the collaborative task is terminated.
Such behaviour is specified by a set of possible states and a set of local possible evolutions on the defined states.

In Tile logic, component states, called configurations in tile logic, are arrows of a horizontal category and are defined as tuples of objects values that the component manipulates. Actions (observations) on component objects are arrows of the vertical category in tile logic. These categories define basic structure of the component and constitute its possible states and actions.

In our collaborative session, identified objects for the president are: *Ports* (a pair of in/out ports is associated to each participant communicating with the president), *List-Participant* (each entry in the list corresponds to a participant and indicates its state connected or disconnected), *Buffer* (contains received messages), *Msg* (all message kinds to send: *msg-invite, msg-open*, *msg-close*, etc.).

These objects are defined by a given signature which is omitted for a simplicity reason.

Configurations correspond to the identified states of the president component. Each configuration ***Cpr*** is a 4-uplet ***(P, L, B, M)***, of objects values where:

P: is a product of n participants ports / *n* is the number of participants.
L : is a list of n communicating participants
B : Buffer of received messages
M : Msg

Basic configurations corresponding to the president possible states are represented as follows:
Pr-Ready = (empty-P, init-L, empty-B, empty-M),

Wait-Response = (Outs , init-L , empty-B, msg-invite)
/ The president has sent an msg-invite and is waiting for a response.

Open-Session = (Outs, L, empty-B, msg-open)
/ The president has sent an msg-open message to all participants (Outs term of the configuration).

Session-Management = (P , L , B , M)
/ The president is realizing the collaborative task with the n participants.

End-Session = (Outs, L, B, msg-close)
/ An msg-close message is broadcast to all participants.

Inviting is an other configuration which is due to an internal action (not visible from the outside) and does not involve an interaction.
We note an n-uplet of all the president outputs as:

$$Outs = \prod_{i=1}^{n} (out(port(i))$$

Since we are more interested here with coordination aspects, we present only a subset of observations that intervene in coordinating tasks of the different components:

Deposit(x:msg, y:port) / deposit a message on all (out)ports
Send(x:port) / sends a content of an out(port).

$Send - all = \bigotimes_{i=1}^{n} send\,(port(i))$ */ corresponds to broadcast action of the deposited message to all participants.*

Receive(i) (x:ports, y:Msg) / indicates a message receipt on the ith *port.*

Consume(i)(x:ports, y:list) / a response withdrawal from port i and an update of participant-list.

As we have already done for the president, we define in a similar manner horizontal and vertical categories for a participant. This type of components manipulates a set of objects: *Status* (*Disconnected, Invited*, and *Connected*), a pair of *Ports* (in/out), a *Buffer* and *Message*.
Each participant configuration is a 4-uplet: ***Cpar* = (*St, P, B, M*).**

Basic configurations of the participant component are: *Par-Ready, Invited, Expected, Connected, Disconnected.*

Possible observations may be:
Receive-Invite(x:port), Receive-Open(x: port), Receive-Close (x:port), Send-Accept (x:port), Send-Reject(x:port), change-st(initial-st: status, final-st : status).

D. Component internal behavior

Possible local evolutions of a component are defined by a set of tiles, controlling components evolution and showing how its state can change when being in given state. Initial and final configurations of a tile correspond to source and target states.
The expected behaviour of the president component is specified as follows:

Prepare: $Pr - \mathrm{Re}\,ady \xrightarrow[deposit(msg-invite,outs)]{id} Inviting$

Being in a *Ready* state (the initial state of the president), the president decides to deposit an inviting message on all its output ports in order to broadcast it to all participants. As a result, the president state topples to *inviting*.

Announce: $Inviting \xrightarrow[Send-all(Outs)]{id} Wait - Response$

Announcement process is terminated by the *msg-invite* broadcast to all participants by executing *Announce* tile. *Announce* tile. Then, the president waits, defined by *Wait-Response* state, for at list a positive response.

Receive-Response:
$Wait - Response \xrightarrow[consume(in(port(i)),l);deposit(msg-open,outs)]{receive(port(i),msg-Accept)} Open - session$

As soon as the president receives an *msg-Accept* on one of hiss input ports (trigger of the *Receive-Response* tile), he deposits an *msg-open* on his output ports in order to declare the session opened.

Inform:
$Open - session \xrightarrow[Send-all(Outs)]{deposit(msg-Open,outs)} Session - Management$

In the same manner as in the announcement process, the *msg-open* is broadcasted to all participants and president becomes in a Session-Management state preparing him self to manage the collaborative session.

Close: $Session - Management \xrightarrow[Send-all(Outs)]{deposit(msg-close,Outs)} End - Session$

The *close* tile is executed when the collaborative task is terminated and the president decides to close it.

In a similar manner, the participant expected behaviour is specified by the following set of tiles. The initial configuration of the tile corresponds to the participant state, while the final configuration corresponds to final one. Each tile is labelled by trigger and an effect:

Receive-Invite:
$Par - \mathrm{Re}\,ady \xrightarrow[change-st(\mathrm{Re}\,ady,Invited)]{\mathrm{Re}\,ceive-Invite(in(p))} Invited$

Prepare-Accept:

$$Invited \xrightarrow[deposit(msg-Accept,out(p))]{id} Invited$$

***Prepare-Reject**:*

$$Invited \xrightarrow[deposit(msg-\mathrm{Re}\,ject,out(p))]{id} Invited$$

Accept:

$$Invited \xrightarrow[Send(out(p))\otimes Change-St(Invited,Expected)]{deposit(msg-Accept,out(p))} Expected$$

Reject:

$$Invited \xrightarrow[Send(out(p))\otimes Change-St(Invited,Disconnected)]{deposit(msg-\mathrm{Re}\,ject,out(p))} Disconnected$$

Receive-Open:

$$Expected \xrightarrow[change-St(Expected,connected)]{\mathrm{Re}\,ceive-Open(in(p))} Connected$$

Receive-Close:

$$Connected \xrightarrow[Change-St(Connected,Disconnected)]{\mathrm{Re}\,ceive-Close(in(p))} Disconnected$$

Up till now and thanks to tile logic, we have formally specified the structure and the behaviour of each component in the software architecture. Similar results could be gained by using other formalisms. The most important aspect is the synchronization of interactions between components.

E. Components Synchronization

In most existing component-based approaches, synchronisation between system components is defined by static connectors, imposing constraints of static interconnection topology definition.. In our model, we have defined a tile logic based dynamic connector. It depends on the contribution of two components to execute a shared action (synchronization), expressed by a tile.

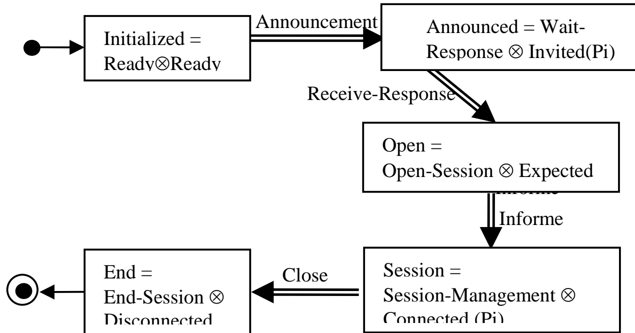

Fig. 3. Session State/Transition Diagram

Announcement for example (see Fig.3), reflects an interaction between the president and a participant component. It corresponds to a synchronization between the sending of the *msg-invite* message by the president and its receipt by the participants. To realisation such synchronization, the following scenario is executed: The president puts an *msg-invite* message in all output ports by executing its *Prepare* tile. Then, he sends the message by executing its *Announce* tile. President state evolves to *Wait-response*. *Announce* tile effect is a broadcast of *msg-invite* to all participants, which actually do nothing (*Par-Ready* state). These states constitute the initial configuration *(Wait-Response ⊗ Par-Ready*) of the synchronization tile (in Fig.4). The resulting configuration is identified by *Wait-Response ⊗ Invited*, since the input port of the corresponding participant contains an *msg-invite* message now. So, the occurrence of a Send observation, as an effect of the president *Announce* tile, triggers the synchronisation tile. It prepares necessary interfaces to the interaction by isolating *out(pr)* and *in(par)* thanks to the parallel composition of horizontal identities id_{Cpr} and id_{Cpar}. Synchronisation tile effect is the following sequence: A connection between the president component and the participant one is realised by duplicating the output port of the former. *Out(pr)* port contents are transferred by swapping one output port copy of the president and the input port of the participant. Then, interaction ends by destroying the empty port. We notice that, duplicator operator ∇ creates a copy of output port and renames it as *in'(par)*.

IV. CONCLUSION

ADLs are, in a way, domain-specific languages for aspects such as coordination, distribution and quality-of-service. Formal theories can be defined, clarifying these concepts or providing rules to determine if a given architecture is well-formed. We have proposed a tile logic based model, where ADLs inherent concepts have been systematically transformed to a formal theory specification, That can be model checked.

Our chosen semantic framework, Tile logic, has been showed as an interesting unified model of all software component aspects through a simple but significant case study, a collaborative application session.

A tile based dynamic connector has also been proposed to define dynamic synchronization between system components.

This explicit exposition has facilitated a natural mapping of a described architecture into a formal software model and can, in a next stage, facilitate dynamic reconfiguration and component mobility activities and non-functional properties analysis.

REFERENCES

[1] N. Medvidovic, R. M. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, Vol 26, no1, pp70-93, Janvier 2000.

[2] M. Clavel, F. Duran, S. Eker, N. Martı-Oliet, P. Lincoln, J. Meseguer, and J. Quesada. Maude: Specification and Programming in Rewriting Logic. SRI International, http://maude.csl.sri.com, January 1999.

[3] C. Braga1, A. Sztajnberg, "Towards a Rewriting Semantics for a Software Architecture Description Language", in: A. Cavalcanti and P. Machado, editors, Proceedings of WMF 2003, 6th Workshop on Formal Methods, Campina Grande, Brazil, E.N.T.C.S. 95 (2003), p.148-168.

[4] R. Bruni, J. L. Fiadeiro, I. Lanese, A. Lopes, and U. Montanari, “New Insight into the Algebraic Properties of Architectural Connectors”, IFIP TCS, pp. 367-380, 2004.

[5] C.Bouanaka, A. Choutri, F. Belala, “On Generating Tile System for a Software Architecture : Case of a Collaborative Application Session”, in ICSOFT2007 (the Second Conference on Software and Data Technologies), pp. 123-128m July 22-25, 2007.

[6] R. Bruni, “Tile Logic for Synchronized Rewriting of Concurrent Systems”, Phd Thesis, University of Pisa, TD-1/99, March 1999.

[7] J. Meseguer, “Conditional Rewriting Logic as a unified model of concurrency”, Theoretical Computer Science, 1992, pp.73-155.

[8] J. M. Molina Espinosa, “Modèles et services pour la coordination des sessions coopératives multi applications: application à l’ingénierie systèmes distribués”, Thèse de doctorat en Informatique et télécommunications, LAAS of CNRS, Toulouse, 2003.

[9] R. Bruni, I. Lanesse, and U. Montanari, “A Basic Algebra of Stateless Connectors”, Theoretical Computer Science 366, pp. 98-120, 2006.