ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2006, Vol.35, No.3

EXTENSION OF PLA SPECIFICATION FOR DYNAMIC SYSTEM FORMALIZATION

Šarūnas Packevičius, Algirdas Kazla, Henrikas Pranevičius

Kaunas University of Technology, Faculty of Informatics
Studentų str. 50, LT - 51368 Kaunas

Abstract. In this paper, an extension of Piece Linear Aggregates (PLA) formalization language is presented. This extension, called dynPLA (dynamic PLA) is intended for formalization of dynamic systems. Dynamic systems are characterized as the ones that change their structure and/or behavior during runtime. This paper covers the extension of PLA – dynPLA model, its specification and meta-model. An example of transaction coordinating systems is given to illustrate presented method.

Keywords: dynamic formal specifications, PLA, DEVS.

1. Introduction

It's hard to imagine today's enterprise without internet site and without information system within enterprise, which supports its processes. More and more systems are created to support e-commerce, e-work, e-health etc. These systems render obsolete if are unable to function in changing environment [5]. Regardless the domain these systems belong to – business, government or health care – in some way they represent the domain itself, which is dynamic [8]. Systems must be able to adapt to volatile environment, e.g. start communicating with new information system of new enterprises partner, accept data in different format from new source etc. System should be able to change its behavior and structure [11] – i.e. adapt. More and more Web services, which can be implemented as agent systems, are used to for systems interoperability. Today, agent applications become more popular, which in turn require systems to be dynamic, able to change behavior and structure at runtime [10, 11]. Complex information systems, such as electronic health record systems (EHRS) from healthcare infrastructure, require enormous flexibility to support various processes of the domain [5]. There are also other types of systems, which in one way or another are required to support change of structure or behavior, e.g. network protocols, networks of systems [1].

As for every system, creation of adaptive systems can be supported by their formal specification, validation, verification and simulation. Formalization and modeling languages are altered to support descriptions of dynamic systems. Dynamic DEVS [11] extension was introduced for popular formalization language DEVS [12]. This extension enabled specification of self modification (structure and behavior). There are also suggestions to use some new languages, such as "Visual Language with Dynamic Specification" [7], M-trans [4], designed to describe adaptive systems.

In this paper, we present an extension of formalization language Piece Linear Aggregates (PLA) [9], supporting dynamics, which will enable to change formal specification of systems and herewith system's structure and behavior. The rest of the paper is organized into sections as follows. Section 2 gives a brief description of PLA formalization language and introduces Dynamic PLA – extensions to specification and its meta-model. Section 3 illustrates dynPLA extension with example. Conclusions are given in Section 4.

2. Dynamic PLA model (dynPLA)

2.1. Extension of PLA model

Dynamic PLA extension requires defining system model, which can change in time. For this, we propose to add a set of aggregates, named A with index t. Elements of set A_t represent system's aggregates at time t. That is, the set A represents the structure of systems model at some specific time.

We suggest to use matrix M with index t, which would hold connections of the system between aggregates at certain time t. An example of the connection matrix M is given at the Table 1.

Also, we suggest the modifications of aggregates description by:

1. Creating set *H* with index *t*, which contains aggregate operators H at given time *t*, and operations with this set, which deletes or adds H operators to this set.
2. Creating set *G* with index *t*, which contains aggregate operators G at given time *t*, and operations with this set, which deletes or adds G operators to this set.

To extend PLA formalization language, it is needed to introduce new operations. Operations are defined in *H* set for each aggregate – i.e. an aggregate can operate with its own discrete and continuous variables, and its and systems structure. Specification of aggregate is extended with the following operations:

1. Addition of new aggregate to the system model: $A_{t+1} = A_t \cup \{A_{\text{new}}\}$, where A_{new} is an aggregate added to the model. Initial state of new aggregate is based on the parameters defined in aggregate.
2. Removal of an aggregate from the system: $A_{t+1}=A_t \setminus \{A_{\text{old}}\}$, where A_{old} is an aggregate removed from the model.
3. Addition of new input to the aggregate: $X_{t+1}=X_t \cup \{x_{\text{new}}\}$, where x_{new} is an input added to the aggregate.
4. Removal of input from the aggregate: $X_{t+1} = X_t \setminus \{x_{\text{old}}\}$, where x_{old} is an input removed from aggregate.
5. Addition of new output to aggregate: $Y_{t+1} = Y_t \cup \{y_{\text{new}}\}$, where y_{new} is an output added to the aggregate.
6. Removal of output from the aggregate: $Y_{t+1}= Y_t \setminus \{y_{\text{old}}\}$, where y_{old} is an output removed from the aggregate.
7. Creation of new connection between aggregates:

   $M_{t+1} =M_t \cup \{5, A_{\text{from}}, y_{\text{from}}, A_{to}, x_{to}\}$, connection (No. 5) is created by connecting aggregates A_{from} output y_{from} to aggregates A_{to} input x_{to}.

   With this action, the whole new line with connection parameters is added to the connection matrix *M*.
8. Deletion of connection between aggregates: $M_{t+1} = M_t \setminus \{ 5, A_{\text{from}}, y_{\text{from}}, A_{\text{to}}, x_{to} \}$, connection (No. 5) from A_{from} aggregates output y_{from} to A_{to} aggregates input x_{to} is deleted.

   The specified line is deleted from the connection matrix *M*.
9. Creation of G, H operators which process events. $H_{t+1} = H_t \cup \{H(e'_{\text{new}})\}$ is an addition of H operator, which processes event e'_{new}. $G_{t+1} = G_t \cup \{G(e'_{\text{new}})\}$ is an addition of G operator, which processes event e'_{new}.

   Operators, which are added, must be defined beforehand in specification of aggregate, but not included in the main *H* and *G* sets. This inclusion should be preformed in specification logics of other H operators.

   Operators G and H are associated, respectively, with inner and outer events. The ability to modify events requires modifying *G*, *H* sets, respectively. That is, by adding inner or outer events to *G*, *H* sets, operators G, H which process these new events, must be added too.
10. Deletion of the G, H: $H_{t+1}=H_t \setminus \{H(e'_{\text{old}}) \}$, where H is an operator, which processes event e'_{old}; $G_{t+1} = G_t \setminus \{G(e'_{\text{old}})\}$, where G is an operator, which processes event e'_{old}.

    Operators G and H are associated, respectively, with inner and outer events. Ability to modify events requires modifying *G*, *H* sets respectively. That is, by deleting inner or outer events from *G*, *H* sets, operators G, H which process these events, must be deleted too.
11. Creation of inner events in aggregate: $E''_{t+1} = E''_t \cup \{ e''_{\text{new}} \}$, where e''_{new} is an inner event added to the aggregate.
12. Deletion of inner events from the aggregate: $E''_{t+1} = E''_t \setminus \{ e''_{\text{old}} \}$, where e''_{old} is an inner event removed from the aggregate.
13. Creation of outer events in aggregate: $E'_{t+1} = E'_t \cup \{ e'_{\text{new}} \}$, where e'_{new} is an outer event added to the aggregate.

    Creation of outer events requires addition of inputs to the aggregate. Therefore if new input is added, then new outer event must be added, too.
14. Deletion of outer events from the aggregate: $E'_{t+1} = E'_t \setminus \{ e'_{\text{old}} \}$, where e'_{old} is an outer event removed from the aggregate.

    Deletion of outer events requires deletion of inputs from aggregate. Therefore if input is deleted, then outer event must be deleted, too.

Defined operations (from 1 to 11) add or delete elements (aggregates, inputs, outputs, connections, operators, events) to/from aggregate system model. For element modification (i.e. for changing elements properties) we suggest to delete an old element (the one that is changed) and add modified one.

Table 1. Connection matrix of aggregates

Channel No.	From aggregate	Output	To aggregate	Input
1	TC	y_1	RM1	x_1
2	TC	y_2	RM2	x_1
3	RM1	y_1	TC	x_0
4	RM2	y_1	TC	x_1
5	RM1	y_2	Agg0	x_1
6	RM2	y_2	Agg0	x_2
7	Agg0	y_1	TC	x_2

We also defined situations when aggregates operator H modifies self description. In some cases it is needed to modify structure of other aggregate. For this we suggest to use the full name of element, which is composed of the aggregate name, symbol "." (dot, period) and the name of element, e.g. to add new input x_{new} to aggregate Agg0, specification should be: $Agg0.X_{t+1} = Agg0.X_t \cup \{ x_{\text{new}} \}$. (Here Agg0 is the name of the aggregate, X – set of inputs).

2.2. DynPLA meta-model

We shall describe the meta-model of Dynamic PLA using UML graphical notation [2], which in turn is described in MOF notation [6]. Meta model of Dynamic PLA is presented as series of UML class diagrams. The top level diagram presents the highly abstract view Dynamic PLA meta-model. Other diagrams provide a detailed description of elements in highly abstract view diagram. The aggregation relation in diagrams presents the idea that one element of Dynamic PLA specification consists of other elements in specification, inheritance specifies that element generalizes other elements in specification. Mainly, diagrams follow standard UML notations syntax.

Meta-model of Dynamic PLA is presented in three views:

- System meta-model
- Aggregate meta-model
- Operation meta-model

2.2.1. System meta-model

System meta-model presents modeled system from the most abstract point. Modeled system is presented by System class, which in own turn contains models, aggregates and other elements, as depicted in Figure 1.

System class represents a whole modeled system as single entity. It holds only the name of modeled system as its only one attribute, and also contains a collection of models which depict behavior and structure of modeled systems. Purpose of system class is to serve as a container for the whole modeled system.

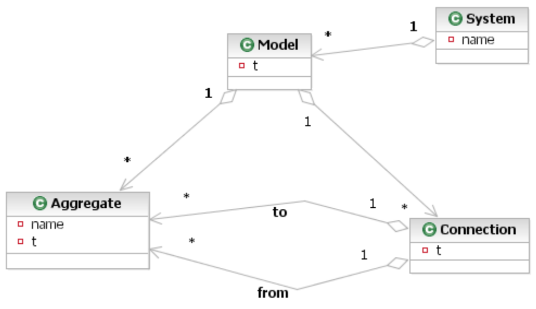

Figure 1. Meta-model of modeled system

Because we consider the dynamic system, model of system can be changed during the existence of the system. So the system is composed of models (i.e. set of models), which represent it at the different time moments. Model at the specified time contains a set of aggregates and connections between them. We assume that the system may contain a finite or infinite number of models, of which one is valid for a specified time frame.

Aggregate class represents aggregate of systems models at the specified time. In system meta-model view, it is depicted as simple class with two attributes: `t` and `name`. The *Aggregate* structure is detailed more in the next subsection. Attribute `t` specifies at which time frame this aggregate is valid.

Connection class represents a connection between two aggregates in the model at the specified time. It has one attribute `t`, which specifies at which time frame this connection is valid. *Connection* joins two aggregates. The set of connections in a model represents a structure of the whole systems at the specified time frame.

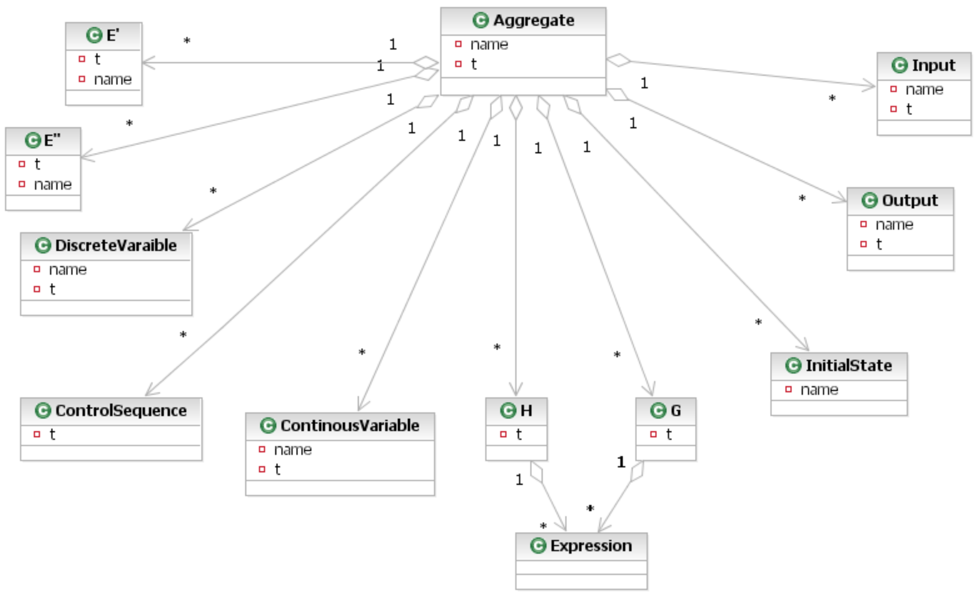

Figure 2. Aggregate meta-model

2.2.2. Aggregate meta-model

Aggregate meta-model presents a detailed view of aggregate in the System meta-model. Aggregate is presented as class which has some attributes and contains other elements as depicted in Figure 2.

Aggregate class represents aggregate of system model at the specified time. It is depicted as complex class with two attributes: `t` and `name`. Attribute `name` specifies aggregates name, and attribute `t` specifies at which time frame this aggregate is valid. Aggregate consists of objects such as: inputs, outputs, initial states, G, H operators, continuous variables, discrete variables, control sequences, *E*", and *E*'. Aggregate embodies a structure of single aggregate at the specified time `t` using inputs, outputs, initial states, continuous variables, discrete variables, control sequences, *E*" and *E*' objects and embodies behavior using G, H operators.

Input class represents input to the aggregate which could be connected by *Connection* object with outputs of other aggregates. Input has `name`, which identifies specific input in concrete aggregate. Input also has `t` attribute which specifies at which time frame this input is valid and thus can be used in connection between two aggregates in a model.

Output class represents output from the aggregate which could be connected by *Connection* object with inputs of other aggregates. Output has `name`, which identifies specific output in concrete aggregate. Output also has `t` attribute which specifies at which time frame this output is valid and thus can be used in connection between two aggregates in a model.

InitialState class represents initial state of the aggregate. Initial state just defines initial values of discrete and continuous variables, which are applied for aggregate creation and addition to the systems model.

E' class represents an external event in the aggregate. It is closely related to input object, because each input must have its representing external event. An external event has the `name` which uniquely identifies this event in the aggregate, and also has `t` attribute which defines at which time frame this event is valid.

E" class represents an internal event in the aggregate. An internal event has the `name` which uniquely identifies this event in the aggregate, and also has `t` attribute which defines at which time frame this event is valid.

DiscreteVariable class represents discrete variable in the aggregate. It has the `name` which uniquely identifies it in the aggregate. It also has the `t` attribute which specifies at which time frame this variable is valid. This variable is used in G an H operators. These operators define aggregates behavior.

ContinousVariable class represents continuous variable in the aggregate. It has the name which uniquely identifies in the aggregate. It also has the `t` attribute which specifies at which time frame this variable is valid. This variable is used in G an H operators. These operators define aggregates behavior.

ControlSequence class represents controlling sequence which controls how some continuous variables change. Control sequence is represented as mathematical expression like function of normal distribution with specific parameters. It has attribute `t` which specifies at which time frame this controlling sequence is valid.

G class represents operators in the aggregate. Operators define some calculations (expressed as mathematical formulas) and produce some results which are outputted to aggregates Outputs. Calculations defined in G operator can be: *Expression*, operations with sets, operations with aggregates and models structure and output operations. G contains `t` attribute which specifies at which time frame this operation is valid.

H class represents operators in the aggregate. Operators define some calculations (expressed as mathematical formulas). Calculations defined in H operator can be: *Expression*, operations with sets, operations with aggregates and models structure. H contains `t` attribute which specifies at which time frame this operation is valid.

Expression class represents a mathematical expression, which performs some calculations, operates with structure of aggregate and/or system model.

2.2.3. Operation meta-model

Operation meta-model details Expression class which was presented in the Aggregate meta-model view. Aggregates expression depictures a composite design patterns [3], it contains either one mathematical expression or a set of them. It can also be an empty set – no operations performed at all. Besides mathematical operations, expression can also be the one which performs modifications of aggregate and/or model structure. This is the core of Dynamic PLA - aggregate can modify its own structure or whole system, thus causing a change of the systems model. Expressions detailed view is presented in Figure 3. Inheritance specifies a new type of operation which perform structure and behavior modification of aggregates or systems.

Expressions AddOutput, AddInput, AddDiscreteVariable, AddContinuousVariable, AddControlSequence, AddE', AddE", AddG, AddH add respectively *Output*, *Input*, *DiscreteVariable*, *ContinousVariable*, *ControlSequence*, *E'*, *E"*, *G*, *H* to the structure of aggregate.

Expressions DelOutput, DelInput, DelDiscreteVariable, DelContinuousVaraible, DelControlSequence, DelE', DelE", DelG, DelH delete respectively Output, Input, DiscreteVariable, ContinousVariable, ControlSequence, E', E", G, H from the structure of aggregate.

Expressions *AddConnecton*, *AddAggregate* add respectively *Connection*, *Aggregate* to the structure of system.

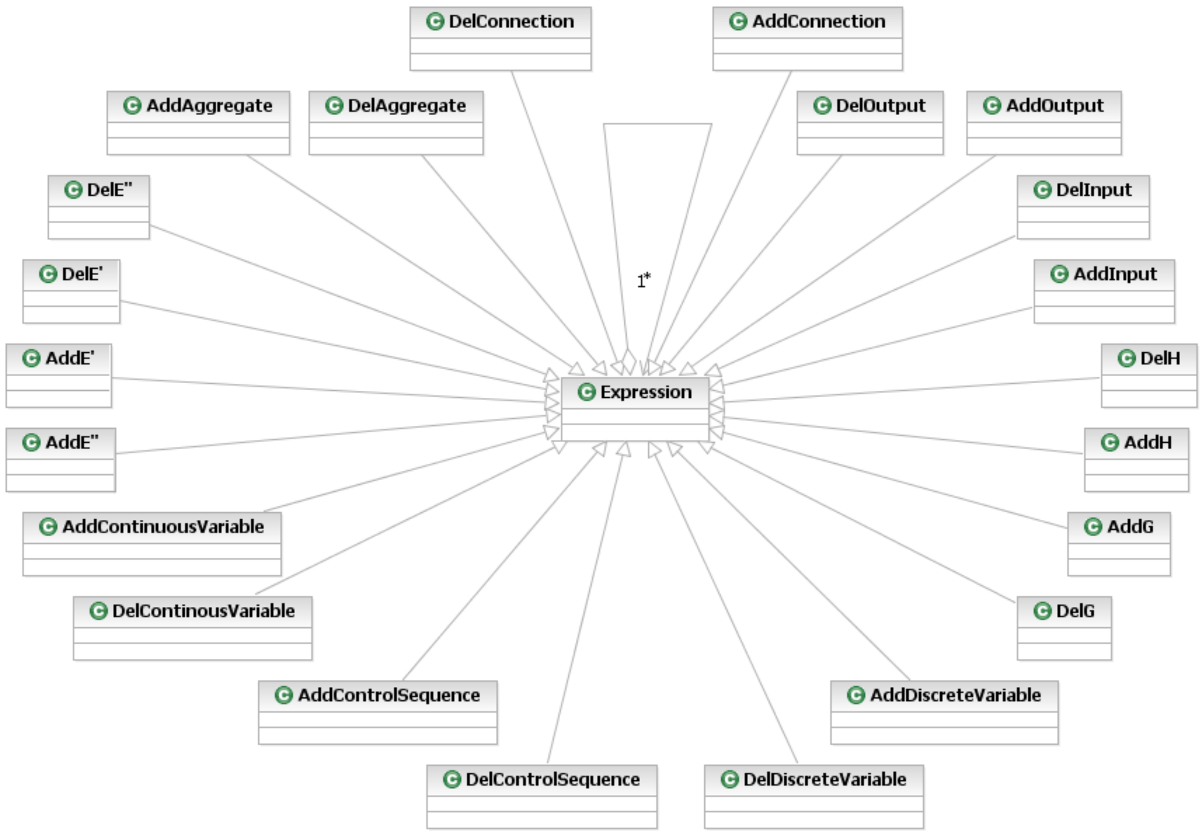

Figure 3. Detailed expressions

Expressions *DelAggregate*, *DelConnection* delete respectively *Connection*, *Aggregate* from the structure of system.

Each Add/Del expression changes model of the system. These expressions depict the dynamic behavior of Dynamic PLA model.

3. Example

3.1. Definition of the resource management system

To illustrate extension of PLA formalization, we present an example – specification of transaction processing system model. The system consists of transaction coordinator (TM), which handles resources (RM), which in turn perform some actions. When transaction coordinator receives a request to perform a task, it allocates a resource and passes the request to it. When resource completes the task, it notifies the transactions manager, which in turn can then free up the resource.

3.2. PLA specification of the resource management system

In order to formalize the system presented in Section 3.1. the PLA model could be built. The systems model is presented in Figure 4 and its PLA specification in Figure 5. The systems model consists of two aggregates: TC and RM_visi where:

- TC – is transaction coordinator, which receives requests to perform tasks. It forwards them to the aggregate called RM_visi (which emulates a dynamically expanding pool of resources which in turn execute requested tasks). TC also receives an acknowledgment of finished tasks from aggregate RM_visi.
- RM_visi – emulates the expanding pool of resource managers. When RM_visi receives a request to perform a task, it creates a resource manager (i.e. new aggregate) which would execute the task. But because the model is static, all tasks are queued and executed one by one in RM_visi. When new task arrives, it is placed in the queue. When RM_visi is ready (all tasks before current task are finished), tasks is taken from the queue and is executed. After completion the task is removed from the queue and the corresponding output is being sent to the TC aggregate.

A formal specification of this system in PLA language is presented in Figure 5.

Using the PLA formalism, the dynamically expanding pool of resource managers, which can perform tasks simultaneously, is emulated by simple queue in aggregate, simplifying simulations of concurrent execution to sequential execution of tasks placed in the queue. In this way, the model does not really match real situation and, in addition, its real nature is hidden by formal specification syntax and models. In general, we can not easily see the dynamic nature of this system (the ability to create a resource manager on request and destroy it after usage). In order to overcome these limitations the systems model is presented using DynPLA in Section 3.3.

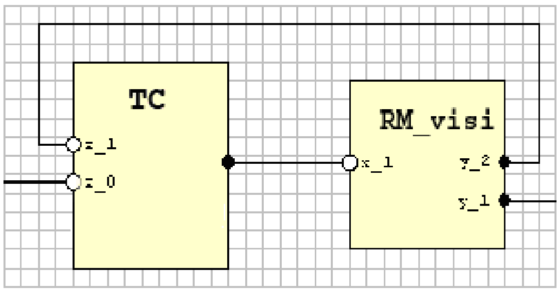

Figure 4. Systems aggregate model

```
TID – transaction ID

TC:
1. X = { $x_0$, $x_1$ } $x_1$ : TID
2. Y = { $y_1$ }
3. E' = { $e'_0$, $e'_1$, }   here $e'_i$ means `received $x_i$ signal`
4. E" = { Ø }
5. Ø
6. $W(t_m)$ = { Ø }
7. $v(t_m)$ = { Ø }
8. Ø
9. H($e'_0$) : /* task request to TC . */
H($e'_i$): /* signal about finished task from resource  */
G($e'_0$):
  Y := { $y_1$ }

$M_t$ =
```

Channel No.	From aggregate	Output	To aggregate	Input
1	TC	y_1	RM_VISI	x_1
2	RM_VISI	y_1	Agg0	x_0
3	RM_VISI	y_2	TC	x_1
4	Agg0	y_1	TC	x_0

```
RM_VISI:
1. X = { $x_1$ }
2. Y = { $y_1$, $y_2$ } $y_2$ : TID
3. E' = { $e'_1$ }
4. E" = { $e''_1$ } $e''_1$ end of operation processing
5. { $e''_1$ } → { $\eta^{(1)}_k$ }, k = 1, ∞
6. $W(t_m)$ = { $w(e''_1, t_m)$ }
7. $v(t_m)$ = ( active_op )
8. active_op = 0
9. H($e'_1$) :
  $active\_opt^{+1} = active\_op_t + 1$
  // adding to queue etc..
H($e''_1$) : // end of processing
  $active\_opt^{+1} = active\_op_t - 1$
G($e'_1$) :
G($e''_1$) : // end of prcessing
  $y_2 = active\_op_t$
  Y : = { $y_1$, $y_2$ }
```

Figure 5. Systems formal specification in PLA language

3.3. Formalization with dynPLA

Changes of analyzed systems model are presented in Figure 6. At the initial moment (t_0) the system has only one aggregate – transaction coordinator (TC, Figure 6a). Later, say, at a time t_n a request comes to transaction coordinator TC, which creates the first resource RM1 with connections (systems model changes – expands) and passes the task (Figure 6b). At some point later (t_m) two more requests arrive at TC, and two more resources (RM2 and RM3) are allocated (Figure 6c). Figure 6d represents system at a time t_p when both resources RM1 and RM2 have already finished their tasks and were deleted from the model (model changes – shrinks).

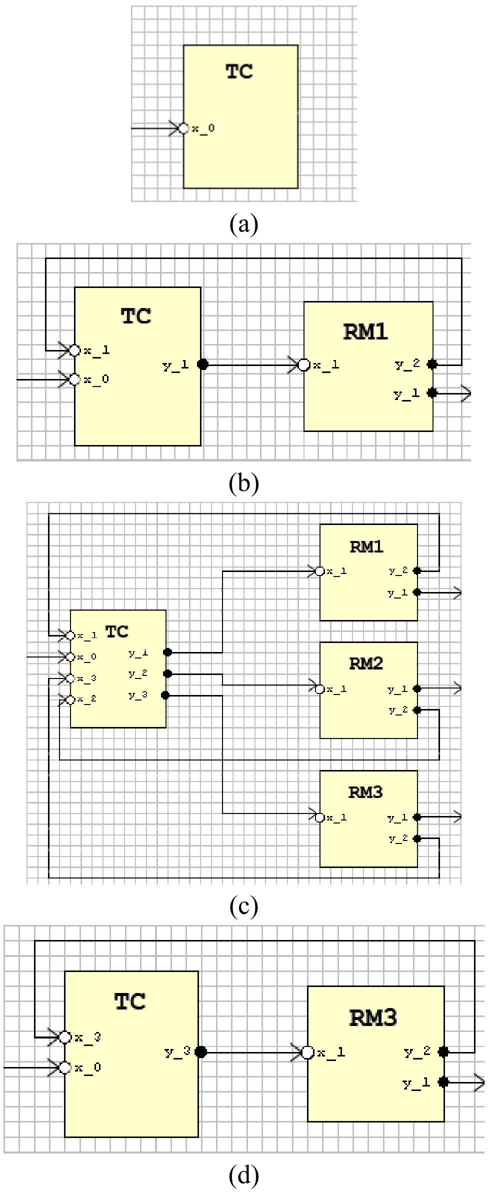

Figure 6. Changes of systems model in time (a) t_0 – initial scheme of the model; (b) t_n – model scheme with added new aggregate (RM1); (c) t_m – added one more aggregate (RM3), (d) t_p - aggregates RM1 and RM2 are removed as they have finished tasks

Specification of this system in dynamic PLA is given in Figure 7.

From standard PLA specification, dynamic PLA differs as follows. In line L1, we define a set of aggregate in the system at initial time t_0. Later this set (i.e. the model of system) will change. Lines L4, L5, L47 and L48 introduce sets of *G* and *H*, respectively, which define G and H operators of aggregates. Lines L16-L23 and L25-L32 introduce dynamic part. Aggregate operators H manipulate the structure of the system. Line L16 adds new aggregate and line L25 removes it. In line L17, new input is added, and in line L27 it is removed.

```
L1   $A_{t0}$ = {TC}
L2
L3   TC:
L4   $H_{t0}$ = { H(e'$_0$), H(e'$_i$) }
L5   $G_{t0}$ = { G(e'$_0$) }
L6   1. X = { $x_0$, $x_1$, $x_2$ }
L7   2. Y = { $y_1$, $y_2$ }
L8   3. E' = { e'$_0$, e'$_1$, e'$_2$}   here e'$_i$ means `received $x_i$ signal`
L9   4. E" = { Ø }
L10  5. Ø
L11  6. $W(t_m)$ = { Ø }
L12  7. $v(t_m)$ = { cnt($t_m$) }
       here cnt($t_m$) – count of aggregates RM in system
L13  8. cnt($t_m$) = 0
L14  9. H(e'$_0$) : /* task request to TC . */
L15    cnt(t+1) = cnt(t) + 1;
L16    $A_{t+1}$ = $A_t$ U { $A_{cnt(t+1)}$ }
L17    $X_{t+1}$ = $X_t$ U { $x_{cnt(t+1)}$ }
L18    $Y_{t+1}$ = $Y_t$ U { $y_{cnt(t+1)}$}
L19    $H_{t+1}$ = $H_t$ U { $H_{cnt(t+1)}$ }
L20    Agg0.$X_{t+1}$ = Agg0.$X_t$ U { $x_{cnt(t+1)}$}
L21    $M_{t+1}$ = $M_t$ U { cnt(t+1)*3–1, TC, $y_{cnt(t+1)}$, $RM_{cnt(t+1)}$, $x_1$ }
L22    $M_{t+1}$ = $M_t$ U { cnt(t+1)*3+0, $RM_{cnt(t+1)}$, $y_2$, TC, $x_{cnt(t+1)}$ }
L23    $M_{t+1}$ = $M_t$ U { cnt(t+1)*3+1, $RM_{cnt(t+1)}$, $y_1$, Agg0, $x_{cnt(t+1)}$ }
L24  H(e'$_i$): /* signal about finished task from resource  */
L25    $A_{t+1}$ = $A_t$ – { $A_{cnt(t+1)}$ }
L26    $X_{t+1}$ = $X_t$ – { $x_{cnt(t+1)}$ }
L27    $Y_{t+1}$ = $Y_t$ – { $y_{cnt(t+1)}$ }
L28    $H_{t+1}$ = $H_t$ – { $H_{cnt(t+1)}$}
L29    $M_{t+1}$ = $M_t$ – { cnt(t+1)*3–1, TC, $y_{cnt(t+1)}$, $RM_{cnt(t+1)}$, $x_1$ }
L30    $M_{t+1}$ = $M_t$ – { cnt(t+1)*3+0, $RM_{cnt(t+1)}$, $y_2$, TC, $x_{cnt(t+1)}$ }
L31    $M_{t+1}$ = $M_t$ – { cnt(t+1)*3+1, $RM_{cnt(t+1)}$, $y_1$, Agg0, $x_{cnt(t+1)}$ }
L32    Agg0.$X_{t+1}$ = Agg0.$X_t$ – { $x_{cnt(t+1)}$}
L33    cnt(t+1) = cnt(t) – 1
L34  G(e'$_0$):
L35    Y := { $y_{cnt(t+1)}$}
L36

L37  $M_{t0}$ =
```

	Channel No.	From aggregate	Output	To aggregate	Input
L38	Channel No.	From aggregate	Output	To aggregate	Input
L39	1	TC	y_1	RM1	x_1
L40	2	TC	y_2	RM2	x_1
L41	3	RM1	y_1	TC	x_0
L42	4	RM2	y_1	TC	x_1
L43	5	RM1	y_2	Agg0	x_1
L44	6	RM2	y_2	Agg0	x_2
	7	Agg0	y_1	TC	x_2
L45					

```
L46  $RM_i$:  // i > 1
L47  $H_{t0}$ = { H(e'$_1$) }
L48  $G_{t0}$ = { G(e'$_1$) }
L49  1. X = { $x_1$ }
L50  2. Y = { $y_1$, $y_2$ }
L51  3. E' = { e'$_1$ }
L52  4. E" = { e"$_1$ }
L53  5. { e"$_1$ } → { $\eta^{(1)}_k$ }, k = 1, ∞
L54  6. $W(t_m)$ = { $w$(e"$_1$ , $t_m$) }
L55  7. $v(t_m)$ = Ø
L56  8. Ø
L57  9. H(e'$_1$) :
L58    /* task processing */
L59  G(e"$_1$) :
L60    Y : = { $y_1$, $y_2$ }
```

Figure 7. Specification of transaction system in dynPLA

In lines L18 and L27, output is added and removed respectively. In lines L19 and L28, operator H is added and removed. It's worth noting that lines L20 and L32 change the structure of another aggregate (input to aggregate Agg0 is added and removed, respectively). Lines L20-L23 create connections between aggregates. Action of adding line, which defines a connection between aggregates, to a set (i.e. $M_{t+1} = M_t$ ∪ {cnt(t+1)*3–1, TC, $y_{\mathrm{cnt}(t+1)}$, $\mathrm{RM}_{\mathrm{cnt}(t+1)}$, x_1}) means that this line is added to the connection matrix *M*. In lines L29-L31, connections are deleted.

4. Conclusions

In the paper we presented extension of PLA formalization to dynamic PLA (DynPLA), adapting this language to define dynamic systems, which can change self structure and/or behavior. Extension DynPLA allows to define model of system more compactly and clearly. However, models of dynamic systems during simulation can change dramatically, which makes harder to notice errors in the initial model specification. This is influenced by addition of new abstraction levels, which is especially noticeable while debugging model.

References

[1] **J. B. Fernando.** Modeling formalisms for dynamic structure systems. *ACM Trans. Model. Comput. Simul., Vol.7*, 1997, 501-515.

[2] **M. Fowler.** UML Distilled: A Brief Guide to the Standard Object Modeling Language. *Third Edition ed. Boston: Addison-Wesley Professional*, 2003.

[3] **E. Gamma.** Design patterns : elements of reusable object-oriented software. *Reading, Mass.: Addison-Wesley*, 1995.

[4] **S. Inoue, M. Iwaihara**. Supporting dynamic process specifications using communication based processes, *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'*02*)-Volume* 9, 2002.

[5] **B. Mario, A.K. Klaus, M. Christian, J. Stefan, L. Richard.** Towards a flexible, process-oriented IT architecture for an integrated healthcare network. *Proceedings of the* 2004 *ACM symposium on Applied computing. Nicosia, Cyprus: ACM Press*, 2004.

[6] **S. J. Mellor**. MDA distilled : principles of model-driven architecture. *Boston: Addison-Wesley*, 2004.

[7] **J. Miyao, C. Shi-Kuo.** A framework of a visual language with dynamic specification, *Proceedings of the* 11*th International IEEE Symposium on Visual Languages* 1995.

[8] **V.P. Nandish.** Adaptive evolutionary information systems. *V.P. Nandish, Ed.: Idea Group Publishing*, 2003, 347.

[9] **H. Pranevičius.** Kompiuterinių tinklų protokolų formalusis specifikavimas ir analizė: agregatinis metodas. *Kaunas: Technologija*, 2003.

[10] **K. Sugawara.** An agent-based framework for developing flexible distributed systems. *Proceedings of the IEEE International Conference on Cognitive Informatics* (*ICCI'*02), 2002.

[11] **A.M. Uhrmacher.** A system theoretic approach to constructing test beds for multi-agent systems. *Discrete event modelng and simulation technologies: a tapestry of systems and AI-based theories and methodologies: Springer-Verlag New York, Inc*., 2001, 315-339.

[12] **B. P. Zeigler, M. Yoonkeon, K. Doohwan, G. Ball.** The DEVS environment for high-performance modeling and simulation. *Computational Science and Engineering, IEEE*, *Vol.*4, 1997, 61-71.

Received May 2006.