ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2009, Vol. 38, No. 1

APPLICATION OF ABSTRACT DATA TYPE IN DYNAMIC PLA APPROACH

Henrikas Pranevičius, Agnė Paulauskaitė-Tarasevičienė, Dalius Makackas

Department of Business Informatics, Kaunas University of Technology
Studentu 56-301, LT-51424 Kaunas, Lithuania
e-mail: agne@ifko.ktu.lt

Abstract. This paper presents the definition of abstract data type (ADT) in dynamic Piece-Linear Aggregate (PLA) model. The introduced ADT permits to describe structural changes in the hierarchical dynamic PLA (dynPLA). In order to formalize the specification of abstract data type, the Z language is used. The application of ADT in specification of dynPLA is demonstrated by an example - the transaction processing system.

Keywords: dynamic systems, formal specification, piece-linear aggregate, abstract data types.

1. Introduction

Recently, there is a need to specify systems, which interact in the dynamic environment and react to various influences by changing the type and the number of their attributes. Such systems are of variable (dynamic) structure, where not only the state but the structure as well is changing in time.

In the real-world applications, there are a lot of activity models, where the tasks related with structural changes are solved. In the biological field, the complex evolution problems are modeled quite often, where dynamic structure is a key feature of such processes [10]. In the simulation of real biological systems a more flexible and understandable description manner is preferable. Such systems usually compose a structure of few levels that leads to the dynamic hierarchical modeling. Multi-agent systems (MAS) are relatively a new research trend, but more and more researches are performed, where agents are used to solve different problems [2]. Specific features of MAS require for various structural changes, such as migration of agents, self-modification and others [4, 5, 6]. There are also other types of systems, which require to support changes of structure, e.g. traffic management, WEB protocols, complex networks [3].

Since such systems usually are of large scale and complex, a motivation for a formal specification occurs. Most of the formal methods can specify only the systems whose structure doesn't vary in time. However, a task of formalization of variable structures is considered in this area as well. For the formalization of dynamic structures, these formalisms need to be extended. There are several formal dynamic approaches, which are based on widely used formal method DEVS (Discrete Event Systems Specification) [16]. In dynDEVS, models are interpreted as a set of different models (incarnations) that are generating themselves by model transition functions [11, 13]. In Multi-level DEVS, apart from variable structures and dynamic ports, multi-level abstraction is provided [14].

Dynamic structure systems are also modeled with various Petri Nets (e.g. dynamic Petri Nets, Colored Petri Nets, high level Petri Nets). For example, the idea of high-level Petri nets is that model can modify its structure by adding/removing places and transitions [15].

Piece-Linear Aggregate (PLA) is a specification formalism based on timed automata [9]. This formal method is used for creation of simulation models and their validation. PLA is very important in designing of complex real time systems, but only of static structure. The extension of PLA – dynPLA has been proposed some time ago [7]. In dynPLA, the specification of aggregate is extended with the new operations, such as addition/removal of new aggregate, addition/removal of new output, and etc. Since dynPLA was presented in the conceptual level not elaborating into the realization details, dynPLA is developed further in this paper. Here, dynPLA is defined using the extended structure of an aggregate, enabling the hierarchical structure. An aggregate consists of the regular attributes used in PLA approach [9] and other internal aggregates with their own connections. To define the structural changes for the system of aggregates, abstract data types (ADT) are introduced in dynPLA. Z specification language [8] is used for the formalization of ADT.

The rest of the paper is organized as follows. In Section 2 PLA model is presented including definitions of classical and dynamic approaches. Section 3 provides Z specification of abstract data type for structural changes in dynPLA. Section 4 illustrates an example where ADT in dynPLA specification is used. We finish with concluding remarks in Section 5.

2. Piece-linear aggregate (PLA) model

2.1. Classical PLA

In classical PLA notation, an aggregate can change only its own state, which consists of discrete and continuous components [9].

Definition 1. Aggregate A is a tuple $A = \langle X, Y, E', E'', Z, H, G \rangle$, where

X – *set of input signals;*
Y – *set of output signals;*
E' – *set of external events;*
E'' – *set of internal events;*
$e_i'' \mapsto \xi_1^i, \xi_2^i, \ldots$ – *controlling sequences*
$Z = \langle v, z_v \rangle$ - *the state of aggregate;*
v – *discrete component;*
z_v – *continuous component;*
H – *transition operator* ($H : E' \cup E'' \times Z \to Z$);
G – *output operator* ($G : E' \cup E'' \times Z \to Y$).

The schema of aggregate A is depicted in Figure 1.

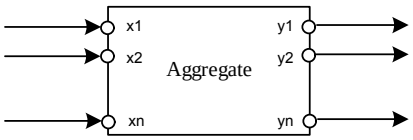


Figure 1. An aggregate

Definition 2. The system of aggregates is a tuple $A_S = \langle \{A_1, A_2, ..A_n\}, R \rangle$, where

– A_i is an aggregate, $A_i = \langle X_i, Y_i, E_i', E_i'', Z_i, H_i, G_i \rangle$ $i = \overline{1, n}$;

– R is the set of links between aggregates of the system

$R = \{A_1, ..A_n\} \times \{Y_1, ..Y_n\} \to \{A_1, ..A_n\} \times \{X_1, ...X_n\}$

The system of aggregates is illustrated in Figure 2.

Usually complex systems are described using a hierarchical approach thereby presenting them as a multi-level structure. Hierarchical structure facilitates the description of tasks which require for the layout of objects in the different levels. To define the hierarchy of aggregates, each aggregate is enabled to have a set of internal aggregates, which in turn can have the connections with their parent aggregate ("surrounding" aggregate) and with each other. All internal aggregates can have a set of aggregates as well. This principle is illustrated in Figure 3. Thus, the structure of hierarchical aggregate consists not only of all attributes of classical PLA (Definition 1) but also includes other connected aggregates (Definition 2), which have the same structure as hierarchical aggregate.

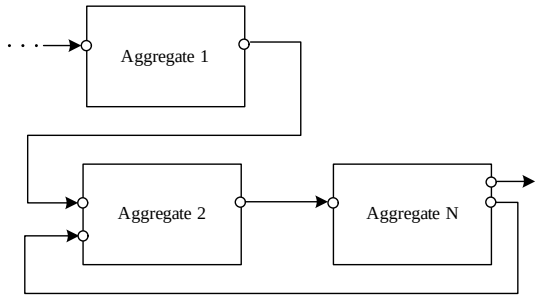


Figure 2. The system of aggregates

Definition 3. Hierarchical aggregate A_h is a structure $A_h = \langle A_0, A_1, ..A_n, R_0, R \rangle$, where

– A_0 is an aggregate, which has a structure described in Definition 1;

– $A_1, ...A_n$ are internal aggregates, which are contained in the aggregate A_0.

– $R_0 = A_0 \times Y_0 \to \{A_1, ..A_n\} \times \{X_1, ..X_n\} \wedge \{A_1, ...A_n\} \times \{Y_1, ...Y_n\} \to A_0 \times X_0$ is the set of links between aggregate A_0 and internal aggregates $A_1, ...A_n$;

– $R = \{A_1, ..A_n\} \times \{Y_1, ..Y_n\} \to \{A_1, ..A_n\} \times \{X_1, ..X_n\}$ is set of links between internal aggregates.

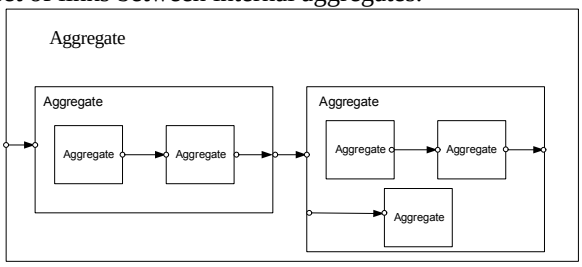


Figure 3. Illustration of hierarchical aggregate

2.2. dynPLA

The purpose of this section is to adapt the hierarchical aggregate to define the systems, whose structure is varying in time. Besides, the aggregates should have a capability to change their own structure autonomously. Each set of aggregate's attributes can be augmented by new element ($X(t_{m+1}) = X(t_m) \cup x_{new}$) or be reduced by removing the existing one ($X(t_{m+1}) = X(t_m) \setminus x_{old}$) at a certain time moment. The changes in the aggregate model are the reactions to the internal and external events. An aggregate can perform the structural modifications, which don't change its interface, since the external changes of the internal aggregate are internal changes of its surrounding one.

Definition 4. dynPLA is the hierarchical PLA model (Def. 3), whose components are varying in time:

$A_{dyn} = \langle X(t), Y(t), E'(t), E''(t), Z\nu(t), \nu(t), H(t), G(t), A_s(t) \rangle$, where:

$X(t)$ – *set of input signals at time moment t;*

$Y(t)$ – *set of output signals at time moment t;*

$E'(t)$ – *set of external events at time moment t;*

$E''(t)$ – *set of internal events at time moment t;*

$Z\nu(t)$ – *set of continuous components at time moment t;*

$\nu(t)$ – *set of discrete components at time moment t;*

$H(t)$ – *set of transition operators at time moment t;*

$G(t)$ – *set of output operators at time moment t ;*

$A_S(t)$ – *system of aggregates at time moment t .*

In classical PLA model, Markov process, which describes the changes of the aggregate's state $z(t)$, includes two components $z_\nu(t)$, $\nu(t)$. Based on dynPLA, the new process is denoted below:

$$z(t) = \langle X(t), Y(t), E'(t), E''(t), \nu(t), Z_\nu(t), H(t), G(t), A_S(t) \rangle,$$

which describes the state of the system at each time moment *t*:

$$z(t) = \begin{cases} z(t_{m-1}), & t \in (t_{m-1}, t_m), \\ z(t_m), & t \in \{t_1, t_2, ..\}, \\ z(t_0), & t = t_0. \end{cases}$$

3. Usage of abstract data type in dynPLA

The main goals of the usage of ADT are: to have clear, precise and unambiguous description of common data with associated operations; to encapsulate the specification; to provide the basis for their realization in programs.

ADT in PLA method was used as well to solve the certain group of problems where the set of particular data with associated operations were used rather frequently in the specification. For instance, ADT of queue was used to specify protocols [9] in order to make the specification more compact and to avoid the declaration of usual operations.

In dynPLA model, four types of structural changes of the system of aggregates may occur: addition of the new link; removal of existing link, addition of the new aggregate, removal of existing aggregate. It is possible to declare the common actions for each group of structural changes described above:

1. Addition of a link: the corresponding link $(A_i, y) \mapsto (A_j, x)$, $i \neq j$ is added to the set of links; new output signal y is added to the set of output signals of source aggregate A_i; new input signal, external event, transition and output operators to process the new signal are added to the corresponding sets of target aggregate A_j.

2. Removal of a link: the link is removed from the set of links; output signal is removed from the set of output signals of source aggregate; input signal, external event, transition and output operators to process the old signal are removed from the corresponding sets of target aggregate.

3. Creation of an aggregate: the aggregate A_i is added to the set of aggregates.

4. Removal of an aggregate: the aggregate A_k is removed from the set of aggregates; all links associated with removed aggregate are deleted $\forall r \in R \,|\, (A_k, y) \mapsto (A_j, x)$ or $\forall r \in R \,|\, (A_i, y) \mapsto (A_k, x)$.

It is appropriate to consider the use of data abstractions for such actions, declaring system of aggregates as a set of data with associated operations listed above. To invoke any of these operations, the precise description of format for input information is defined. To add or remove a link, the information of type $(A_i, y) \mapsto (A_j, x)$, $i \neq j$ is required. To add the new aggregate, the name and type of aggregate A_k have to be referred. To remove the certain aggregate A_k, only the name of the aggregate is required, since all names of aggregates are unique.

3.1. Formalization of abstract data types using Z notation

Specifications of abstract data types provide the basis for their realization. Formal specification is used to validate the statement about model description. For the formal description of ADT specification, Z notation was chosen [8]. This method enables an unambiguous description of all actions for structural changes in the system of aggregates. Restrictions and conditions which have to be met in order to perform structural changes correctly and to avoid faults were included, e.g. we can't create a link, which already exists. The specification of abstract data type was validated using Z notation prover Z/EVES, which has possibilities to check the syntax and semantic of models or even to write theorems [12].

3.2. Z specification of abstract data type for structural changes

In Z specification language all data abstractions and operations are defined as separate components called *schemas*. The specification of ADT of structural changes in dynPLA is described below.

Global sets of attributes *[ID, Xp, Yp, E1p, E2p, υp]* are given bellow:

- *ID* – set of all possible names of aggregates;
- *Xp* – set of all possible input signals;
- *Yp* – set of all possible output signals;
- *E1p* – set of all possible external events;
- *E2p* – set of all possible internal events;
- *υp* – set of all possible discrete components.

The operations for creation/deletion of links use the type information *RInformation* as input parameters. *RInformation* includes the name *ID* of source

aggregate with its output signal Yp and the name ID of target aggregate with its input signal Xp:

$$RInformation = (ID \times Xp) \times (ID \times Yp).$$

The operation for creation of new aggregate uses the type information *AgInformation* as input parameters, which includes the name and type of new aggregate:

$$AgInformation = (ID \times Agg).$$

The state of surrounding aggregate is described using state variables of Z schema *Aggregate*.

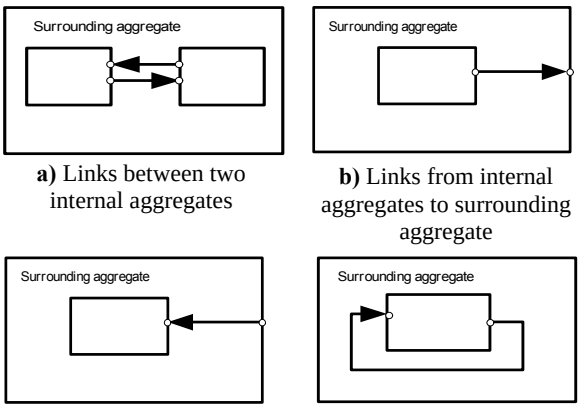
Aggregate

$id: ID$
$X: \mathbb{P}\, Xp$
$Y: \mathbb{P}\, Yp$
$E1: \mathbb{P}\, E1p$
$E2: \mathbb{P}\, E2p$
$\upsilon: \mathbb{P}\, \upsilon p$
$Z\upsilon: E2p \rightarrow \mathbb{N}$
$contr\_seq: E2p \rightarrow \xi$
$H: Zp \times Ep \rightarrow Zp$
$G: Zp \times Ep \rightarrow Yp$
$R: ID \times Xp \rightarrow ID \times Yp$
$Ag: ID \rightarrow Agg$

---

dom $contr\_seq = E2$
$E2 =$ dom $Z\upsilon$
$\forall r: (ID \times Xp) \times (ID \times Yp) \mid r \in R$
• $\exists a1, a2: ID; x: Xp; y: Yp; i, j: Agg$
| $r = (a1, x) \mapsto (a2, y) \wedge i \in \text{ran}\, Ag \wedge j \in \text{ran}\, Ag$
• $a1 = i\,.\,id1 \wedge a2 = j\,.\,id1 \wedge x \in i\,.\,X1 \wedge y \in j\,.\,Y1$
$\vee\ a1 = i\,.\,id1 \wedge a2 = id \wedge x \in i\,.\,X1 \wedge y \in Y$
$\vee\ a1 = id \wedge a2 = j\,.\,id1 \wedge x \in X \wedge y \in j\,.\,Y1$
$\vee\ a1 = i.id1 \wedge a2 = i\,.\,id1 \wedge x \in i.X1 \wedge y \in i\,.\,Y1$

The set Ag of internal aggregates is of the structure $Agg \triangleq [id : ID, X : \mathrm{P}Xp, Y : \mathrm{P}Yp, \ldots]$, which has the same structure as Z schema *Aggregate*.



a) Links between two internal aggregates

b) Links from internal aggregates to surrounding aggregate

c) Links from surrounding aggregate to internal aggregates

d) Links from internal aggregate to itself

Figure 4. Allowed types of links in dynPLA

In the predicate of schema *Aggregate*, the different kinds of possible links are defined: 1) between two internal aggregates (Figure 4a); 2) from internal aggregates to surrounding aggregate (Figure 4b.); 3) from surrounding aggregate to internal aggregates (Figure 4c); 4) from internal aggregate to itself (Figure 4d.).

3.2.1. Operations

Four fundamental schemas for the structural changes of system of aggregates were developed.

Add_R schema describes the creation of the new link. It has two preconditions, which define the constraints on the operation: aggregates *id1*, *id2,* which will be connected by the new link, have to belong to the set of aggregates *Ag*; the new link *newR* can't exist in the set of links *R*. If it is true, the predicate of *Add_R* specifies that the set of links after the completion of the operation is augmented by the new link *newR*. In this operation, all state variables remain unchanged, except *R*.

Add_R

$\Delta Aggregate$
$newR?: RInformation$

$\exists id1, id2: ID; x: Xp; y: Yp \mid newR? = (id1, x) \mapsto (id2, y)$
• $id1 \in \text{dom}\, Ag \wedge id2 \in \text{dom}\, Ag \wedge (id1, x) \mapsto (id2, y) \notin R$
$R' = R \cup \{newR?\}$
... ...

Only the added link is visible for external observer. The aggregates, which have to be connected by the new link, perform structural changes in their inside structure as well. *AgchangesX* and *AgchangesY* schemas define the structural changes in the source and target aggregates.

AgchangesX

ΔAgg
$newR?: RInformation$

$\exists id1, id2: ID; x: Xp; y: Yp; e: Ep; z: Zp \mid newR? = (id1, x) \mapsto (id2, y) \wedge e = x \wedge x \notin IX \wedge e \notin E11$
• $X1' = X1 \cup \{x\}$
$\wedge\ E11' = E11 \cup \{e\}$
$\wedge\ H11' = H11 \cup \{((z, e) \mapsto z)\}$
$\wedge\ G11' = G11 \cup \{((z, e) \mapsto y)\}$
... ...

AgchangesY

ΔAgg
$newR?: RInformation$

$\exists id1, id2: ID; x: Xp; e: Ep; y: Yp; z: Zpp; e: Ep$
$\mid newR? = (id1, x) \mapsto (id2, y) \wedge y \notin Y1$
• $Y1' = Y1 \cup \{y\}$
... ...

In order to apply such actions only to the aggregates, which have to be connected by the new link, the partial operations of *Aggregate* schema named as the *framing schemas* were used.

The first framing schema *ΦFraming1* defines the changes in the source aggregate, the second one *ΦFraming2* in the target aggregate.

ΦFraming1

$\Delta Aggregate$
ΔAgg
$newR?: RInformation$

$\exists id1, id2: ID; x: X; y: Y; i: Ag; type: Agg$
$\mid newR? = (id1, x) \mapsto (id2, y) \land i = id1 \mapsto type$
$\bullet\ id1 \in \mathrm{dom}\ Ag \land \theta Agg\,.\, id = id1 \land Ag' = Ag \setminus \{i\} \cup \{(id1 \mapsto \theta Agg')\}$

ΦFraming2

$\Delta Aggregate$
ΔAgg
$newR?: RInformation$

$\exists id1, id2: ID; x: X; y: Y; i: Ag; type: Agg$
$\mid newR? = (id1, x) \mapsto (id2, y) \land i = id2 \mapsto type$
$\bullet\ id1 \in \mathrm{dom}\ Ag \land \theta Agg\,.\, id = id2 \land Ag' = Ag \setminus \{i\} \cup \{(id2 \mapsto \theta Agg')\}$

Since the framing schema by itself does not represent any system operation, it is combined with previous definitions: ΔAgg, $AgchangesX$ or $AgchangesY$. Above defined framing schemas are combined in the following way:

$$SchemaforX \mathrel{\widehat{=}} \exists \Delta Agg \cdot \Phi Framing1 \land AgchangesX$$

$$SchemaforY \mathrel{\widehat{=}} \exists \Delta Agg \cdot \Phi Framing2 \land AgchangesY$$

Finally, the operation of the link creation can be defined as a composition of *SchemaforX, SchemaforY* and *Add_R* schemas:

$$AddR \mathrel{\widehat{=}} SchemaforX \land SchemaforY \land Add\_R$$

The operation for removal of link is described in the same way as operation of link creation, whereas all corresponding signals (input and output), transition and/or output operators and link are not added but deleted.

Remove_R

$\Delta Aggregate$
$oldR?: RInformation$

$\exists id1, id2: ID; x: Xp; y: Yp \mid oldR? = (id1, x) \mapsto (id2, y)$
$\bullet\ id1 \in \mathrm{dom}\ Ag \land id2 \in \mathrm{dom}\ Ag \land (id1, x) \mapsto (id2, y) \in R$
$R' = R \setminus \{oldR?\}$
... ...

AddAg schema describes the creation of new aggregate. The new aggregate can be added only if it doesn't belong to the set of internal aggregates *Ag*.

AddAg

$\Delta Aggregate$
$newAg?: AgInformation$

$newAg? \notin Ag$
$Ag' = Ag \cup \{newAg?\}$
... ...

RemoveAg schema describes the removal of the aggregate. To remove the aggregate from the system, it is not sufficient to delete the aggregate from the set of aggregates. All links associated with aggregate have to be removed as well.

RemoveAg

$\Delta Aggregate$
$old?: AgInformation$
$oldR!: RInformation2$

$Ag \neq \{\} \land old? \in Ag$
$\exists idnew: ID; type: Agg \mid old? = (idnew, type)$
$\bullet$ **if** $idnew \notin \mathrm{dom}\ (\mathrm{dom}\ R) \land idnew \notin \mathrm{dom}\ (\mathrm{ran}\ R)$
then $Ag' = Ag \setminus \{old?\} \land R' = R$
else $Ag' = Ag \setminus \{old?\}$
$\land (\forall r: R$
$\mid \exists id1, id2: ID; x: Xp; y: Yp \mid r = (id1, x) \mapsto (id2, y)$
$\bullet\ id1 = idnew \lor id2 = idnew \bullet (r \in oldR! \land R' = R \setminus \{r\}))$
$old? \in Ag$
... ...

4. An example

To demonstrate the application of abstract data type in dynPLA, an example – the model of transaction processing system is introduced.

The transaction processing system consists of transaction coordinator (TM), which handles resources (RM), which in turn perform certain actions. When transaction coordinator receives a request to perform a task, it forwards this task to the particular resource. When resource completes the task, it notifies the transaction coordinator, which in turn can free up the resource [7].

4.1. Specification of the transaction processing system

The transaction processing system is presented in Figure 5. The system of aggregates includes such aggregates:

- External aggregate A_0;
- Transaction coordinator TC;
- Resources RM_k.

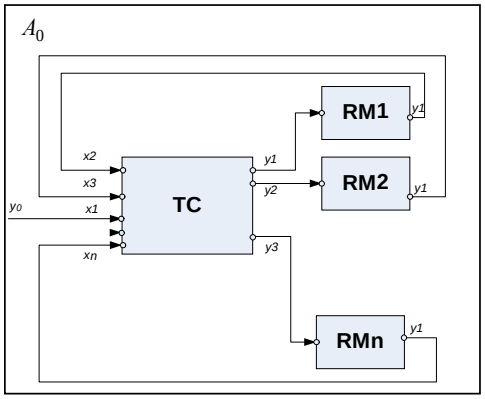


Figure 5. The structure of the system of aggregates

In the specification of transaction processing system given below, ADT (section 3.2) are used to describe the structural changes of the system of aggregates:

- *AddAg* – to create the new aggregates RM_k;

- *RemoveAg* – to remove the existing aggregates RM_k;
- *AddR* – to create new links between TC and aggre-gates RM_k.

Formal descriptions of all aggregates of system are presented using method of controlling sequences [9].

4.1.1. The system of aggregates

The analyzed system is presented as a set of aggregates:

$$A_s(t) = \langle \{A_0(t), TC(t) \cup RM\}, R(t) \rangle,$$

where

$$RM: \| \{RM_1, ...RM_n\},$$
$$R(t) = \{(A_0, y_0) \to (TC, x_1),$$
$$(TC, y_k) \to (RM_k, x_1),$$
$$(RM_k, y_1) \to (TC, x_m)\},$$

where $k = \overline{1,(^{\#}RM)}$, $m = \overline{2,(^{\#}RM+1)}$.

Below is depicted the structure of the system of aggregates at initial time moment (Figure 6).

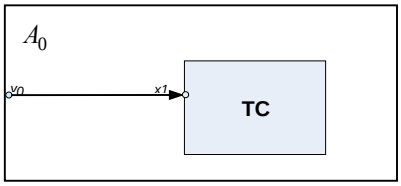


Figure 6. The model of Transaction processing system at initial time moment.

In this time moment, the system has two connected aggregates *TC* and A_0:

$$A_s(t_0) = \langle \{A_0(t_0), TC(t_0)\}, R(t_0) \rangle,$$

where $R(t_0) = (A_0, y_0) \to (TC, x_1)$.

4.1.2 A_0 aggregate

Aggregate A_0 is responsible for generation of the new tasks, which are transmitted to the transaction coordinator *TC.*

1. $X = \varnothing$.
2. $Y = \{y_0\}$.
3. $E' = \varnothing$.
4. $E'' = \{e''_1\}$ – the generation of the new tasks.

$\{e''_1\} \to \{\varphi_k^{(1)}\}$, $k = \overline{1,\infty}$, where $\varphi_k^{(1)}$ – the time period between generation of *k*-th and *(k-1)*th tasks.

5 $\nu(t) = \varnothing$.

6 $z_\nu(t) = w(e''_1, t_m)$, $z_\nu(t_0) = \infty$

7. $H = \{H(e''_1)\}$.

8. $G = \{G(e''_1)\}$.

$H(e''_1)$: // the end of generation of the new task//

$$w(e''_1, t_m + 1) = t_m + \eta_i^1,$$

$G(e''_1)$: $y_0 = task$.

4.1.3. *TC* aggregate

TC is a transaction coordinator, which receives requests from external aggregate A_0 and performs the tasks. All received requests are inserted into the FIFO queue. It forwards each request to the corresponding resource aggregates. When a resource completes the task, it notifies the TC aggregate, which in turn release the resource.

1. $X(t) = \{x_1, x_2, ...x_{cnt(t)+1}\}$.

2. $Y(t) = \{y_1, y_2, ...y_{cnt(t)}\}$.

3. $E'(t) = \{e'_1, e'_2, ...e'_{cnt(t)+1}\}$.

4. $E'' = \{e''_1\}$, where e''_1 – is processing of the task

Set E" is not varying in time.

$\{e''_1\} \to \{\eta_k^{(1)}\}$, $k = \overline{1,\infty}$, where $\eta_k^{(1)}$ – the time period between processing of *k*-th and *(k-1)*th tasks.

5. $\nu(t) = \{A_S(t), Q(t), cnt(t)\}$,

where $Q(t)$ – a queue of tasks;.

$A_S(t)$ – a set of aggregates of system;

$cnt(t)$ – the number of active resources.

6. $z_\nu(t) = \{w(e''_1, t_m)\}$.

7. $H(t) = \{H(e'_1), H(e'_2), ...H(e'_{cnt(t)+1})\}$.

8. $G(t) = \{G(e'_1), G(e'_2), G(e'_{cnt(t)+1})\}$.

Descriptions of transition and output operator, which define the changes of RA aggregate's coordinates are presented below

$H(e'_1)$: // the new task is received //

$$ENQ(Q(t_m), task), \quad if \quad w(e''_1, t_m) \neq \infty,$$
$$w(e''_1, t_m) = t_m + \eta_k^1, \quad if \quad {}^{\#}Q(t_m) = 0,$$

$G(e'_1)$: $Y = \varnothing$.

$H(e''_1)$: // the end of processing of the task //

$$\left.\begin{array}{l} DEQ(Q(t_m)), \\ cnt(t_m + 1) = cnt(t_m) + 1, \\ AddAg(A_S(t_m), CA). \end{array}\right\} if \quad {}^{\#}Q(t_m) \neq 0,$$

where $CA = RM_{cnt(t_m+1)}$,

$AddR(A_S(t_m), CR)$,

where $CR = \{(RM_{cnt(t_m+1)}, y_1) \to (TC, x_{cnt(t_m+1)+1}),$

$(TC, y_{cnt(t_m+1)}) \to (RM_{cnt(t_m+1)}, x_1)$,

$$w(e''_1, t_m) = t_m + \eta_k^1, \quad if \quad {}^{\#}Q(t_m) = 0,$$

$G(e''_1)$: y_{1h}, where $h = cnt(t_m + 1)$.

$H(e'_k)$: // the forwarded task is accomplished //

$RemoveAg(A_S(t_m), DA)$, where $DA = RM_k$,

where $k = \overline{2, cnt(t)+1}$,

$G(e'_k)$: $Y = \varnothing$.

The structure of aggregate *TC* at the initial time moment is described below:

1. $X(t_0) = \{x_1\}$,

2. $Y(t_0) = \varnothing$,
3. $E'(t_0) = \{e'_1\}$,
4. $E'' = \{e''_1\}$,
5. $\nu(t_0) = \{A_S(t_0), Q(t_0), cnt(t_0)\}$,

where $Q(t_0) = 0$,

$cnt(t_0) = 0$,

the structure of $A_S(t_0)$ at initial time moment is defined in section 4.1.1.

6. $z_\nu(t_0) = \{\infty\}$,
7. $H(t_0) = \{H(e'_1)\}$,
8. $G(t_0) = \{G(e'_1)\}$.

4.1.4. RM_k aggregate

The aggregate RM_k receives a task from transaction coordinator *TC*. During the internal event, the received task is performed. Whenever the task is accomplished, the resource aggregate RM_k informs transaction coordinator *TC* by generating the output signal.

1. $X = \{x_1\}$.
2. $Y = \{y_1\}$.
3. $E' = \{e'_1\}$.
4. $E'' = \{e''_1\}$ – the performance of received task.

$\{e''_1\} \to \{\varphi_1^{(1)}\}$ – the processing duration of the task.

5 $\nu(t) = \varnothing$.

6 $z_\nu(t) = w(e''_1, t_m)$, $z_\nu(t_0) = \infty$

7. $H = \{H(e'_1), H(e''_1)\}$.
8. $G = \{G(e'_1), G(e''_1)\}$.

$H(e'_1)$: // the forwarded task is received //

$w(e''_1, t_m + 1) = t_m + \varphi_i$

$G(e'_1)$: $Y = \varnothing_1$.

$H(e''_1)$: // the performance of received task //

$w(e''_1, t_m + 1) = \infty$

$G(e''_1)$: $Y = y_1$.

In this example, the structures of aggregates RM_k and A_0 are not varying in time. System of aggregates is used as abstract data type. Defined operations (section 3.2.1) in advance were enough to perform all structural changes in analyzed system.

5. Concluding remarks

In this paper we demonstrated the usage of abstract data type (ADT) for structural changes in the dynamic PLA model. For the formalization of ADT, Z specification language has been chosen, since it provided a complementary representation of the dynamic behavior of aggregates. Besides, Z notation allowed us to define ADT in mathematically rigorous manner based on the set theory and predicate calculus. Introduced ADT has been verificated using Z/EVES prover. It permits to check the syntax and the semantic of specification, ensuring that ADT was defined correctly. In dynPLA, the application of predefined ADT allowed us to get the compact specification for considered example.

In the future, this approach will be used for formalization of the Session Invitation Protocol (SIP).

References

[1] **F. J. Barros**. Modeling formalisms for dynamic structure systems. *ACM Trans. Model. Comput. Simul. Vol*.7, *No.* 4, 1997, 501–515.

[2] **F. Bellifemine, G. Claire, D. Greenwood**. Developing Multi-Agent Systems with JADE. *John Wiley & Sons. Ltd.,* 2007.

[3] **S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D.-U. Hwang**. Complex networks: Structure and dynamics. *Physics Reports*, *Vol.*424*, No.* 4-5*,* 2006, 175–308.

[4] **F. M. T. Brazier, N. J. E. Wijngaards.** Designing Self-Modifying Agents. *Proceedings of Computational and Cognitive Models of Creative Design, the fifth international roundtable conference Computing, University of Sydney*, 2001, 93-112.

[5] **F. M. T. Brazier, B. J. Overeinder, M. van Steen, N. J. E. Wijngaards**. Generative Migration of Agents. *Proceedings of the AISB'*02 *Symposium on Adaptive Agents and Multi-Agent Systems*, 2002, 116-119.

[6] **V. Dignum, F. Dignum, L. Sonenberg**. Towards dynamic organization of agent societies. *Workshop on Coordination in Emergent Agent Societies,* 2004, 70–78.

[7] **Š. Packevičius, A. Kazla, H. Pranevičius.** Extension of PLA Specification for Dynamic System Formalization. *Information Technology And Control,* 2006, *Vol.35, No*.3, 235 – 242.

[8] **B. Potter, J. Sinclair, D. Till** . An Introduction to Formal Specification and Z. *Prentice Hall, Trowbridge, UK*, 1996.

[9] **H. Pranevičius**. Formal Specification and Analysis of Computer Network Protocols: Aggregate Approach. *Technologija, Kauna*s, 2004 *(in Lithuanian)*.

[10] **J. R. Prill, P.A. Iglesias, A. Levchenko**. Dynamic Properties of Network Motifs Contribute to Biological Network Organization. 2005, *PloS Biology*, *Vol*. 3(11), 650 – 659.

[11] **M. Rohl, A. M. Uhrmacher**. Controlled Experimen-tation with Agents - Models and Implementations. *5th International Workshop. "Engineering Societies in the Agents World", Toulouse, France,* 2004, *October*, 20-22, 292-304.

[12] **M. Saaltink**. The Z/EVES 2.0 User's Guide. *TR* – 99-5493-06*a, ORA Canada, Canada,* 1999.

[13] **A. M. Uhrmacher**. Dynamic Structures in Modeling and Simulation : A Reflective Approach. *ACM Transactions on Modeling and Computer Simulation, Vol.11, No.2,* 2001, 206-232.

[14] **A. M. Uhrmacher, R. Ewald, M. John, C. Maus, M. Jeschke, S. Biermann**. Combining Micro and macro-modeling in DEVS for computational biology. *Proceedings of the* 2007 *Winter Simulation Conference, IEEE Press*, 2007, 871-880.

[15] **D. Xu, Y. Deng**. Modeling mobile agent systems with high level Petri nets. *Proc. IEEE International Conference on Systems, Man and Cybernetics* (*SMC'*2000), *Vol.*5, 2000, 3177-3182.

[16] **B. P.Zeigler, H. Praehofer, T. G. Kim**. Theory of Modeling and Simulation. *Second Edition, Academic Press,* 2000.

Received October 2008.