PROGRAM COMPONENTS & ABSTRACT BEHAVIORAL TYPES

Marián JENČIK, Daniel MIHÁLYI
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice,
Letná 9, 042 00 Košice, Slovak Republic, e-mail: marian.jencik@student.tuke.sk, daniel.mihalyi@tuke.sk

ABSTRACT

In this paper we deal with the construction of the component-based complex program system categorical model which will be used to behavioral description. The basis of the behavioral description of the component-based complex system and definition of the model is Abstract Behavioral Type (ABT) *which yields behavior of the program component as maximal interactions among observable timed data streams. Program component is an independent entity, black-box, from which we can compose component-based complex program system and an interaction is in this case defined as a relation between observable program component input and output through component ports. Observable input and output is yielded as a pair of the infinite data stream and timed stream. Complexity of the system is given by number of ABTs where individual complex program system we can divide into two kinds of ABT types, namely component instance's ABT whose internal structure is unknown and connector's ABT whose internal structure is known and can be defined by user. Behavior of the whole complex system can be yielded as a conjunction of the ABTs.*

Keywords: *component-based programming, program component, abstract behavioral type, complex program system*

1. INTRODUCTION

Component-based software engineering, as a part of software engineering deals with the construction of program systems by combining prefabricated components with new programs that provide both glue between the components, and new functionality [2].

Each program component can be defined as follows:

Definition 1.1. *Program component form which is program system composed is by [6] defined as a unit of composition with contractually specified interfaces and fully explicit context dependencies that can be deployed independently and is subject to third party composition.*

A program component have to match three following characteristics [5]:

1. Black-box composability, substitutability and reusability: *there is no need to know the design and the implementation when composing a component with other parts of the system, substituting a component with another one or reusing it in another application.*
2. Independent development: *components can be designed, implemented, verified, validated and deployed independently.*
3. Interoperability: *components can be implemented in different programming languages and paradigms, but they can be composed, be glued together and they cooperate with each other.*

□

In a programming paradigm where instances of components are their primitive buildings blocks, all decisions about which components will be used and composed together to build a new program application must be made from outside what in generally means that a component cannot be allowed to internally decide that component will be communicate with another components.

2. ABSTRACT DATA TYPE (ADT)

If we try to use *Abstract Data Type (ADT)* which is defined in Definition 2.1 and which has served as a foundation structure for structural and object-oriented programming for some decades as a model for program components then only way to communicate with a program component is by invoking it's operations and inter-component communication becomes the same as inter-object communication. This way of inter-component communication would be for program components rather restrictive. A formal model of components has to provide an inter-communication mechanism that affords a higher level of mutual independence to components than the ADT model does.

Definition 2.1. Abstract Data Type (ADT) *is formally defined as a triple*

$$(\mathscr{T}, \mathscr{F}, \mathscr{A}), \tag{1}$$

where $\mathscr{T}$ *is a set of* sorts *denoting the required types,* $\mathscr{F}$ *is a set of* operations *over* $\mathscr{T}$ *and* $\mathscr{A}$ *is a set of* axioms *written as algebraic equations defining the results of various combinations of operations in* $\mathscr{F}$ *on data items of various types in* $\mathscr{T}$ *.*

□

Abstract Behavioral Type (ABT) as a model for program components in contrast to ADT supports only sending untargeted passive message through it's contact ports. Components exchanges with it's environment some passive data :

- *sending message* is just write-operation and
- *receiving message* is just read-operation.

This view of inter-component communication for component-based system is more appropriate.

ISSN 1335-8243 (print)
www.aei.tuke.sk

ISSN 1338-3957 (online)
www.versita.com/aei

3. ABSTRACT BEHAVIOR TYPE (ABT)

Abstract Behavioral Type (ABT) in contrast of abstract data type (ADT) which contains operations and data structures that may be used by operational interface implementation defines behavior of the program component based on observable program component input and output .

Definition 3.1. *An* Abstract Behavior Type *is an interaction between observable input and output that occur through component ports without specifying any detail about:*

1. *the operations that may be used to implement such behavior or*
2. *the data types those operations may manipulate for the realization of that behavior.*

The set of component ports we can name interface *of the component.*

□

4. INTERACTION

If we can define an interaction between observable input and output of the component which can be denoted as :

$$\text{input}_i \xrightarrow{\text{interaction}_k} \text{output}_j,$$

where input_i is observable input and output_j is observable output, for $0 < i \leq m$ a $0 < j \leq n$,

we have firstly to define *Data Stream (DS)* and *Timed Data Stream (TDS)*.

Definition 4.1. Data Stream (DS) *is defined as an infinite sequence of elements over set of data items and is denoted* $\mathscr{D}^{\omega}$. *Stream over data items is very often denoted by characters of the Greek alphabet as* α, β, γ, ….

□

The elements of an individual data stream are by [7] numbered as the first, the second, the third, etc. elements of the data stream, e.g., $\alpha(0)$, $\alpha(1)$, $\alpha(2)$, $\alpha(3)$, … where $\alpha(0)$ we call *initial value* of the data stream α.

Definition 4.2. Timed Data Stream (TDS) *is defined as a pair of streams*

$$\langle \alpha, a \rangle, \tag{2}$$

where $TDS = \mathscr{D}^{\omega} \times \mathscr{T}^{\omega}$ *consists of data stream,* $\alpha \in DS$ *and timed stream,* $a \in TS$, *with the interpretation that an input resp. an output of a data item* $\alpha(i)$ *occurs at* time moment $a(i)$, *for all* $i \geq 0$.

□

Notation 4.1. *Two* timed data streams $\langle \alpha, a \rangle$ *and* $\langle \beta, b \rangle$ *are equal if their respective elements are equal*

$$\langle \alpha, a \rangle = \langle \beta, b \rangle \equiv \alpha = \beta \wedge a = b. \tag{3}$$

□

An interaction between program components, which are independent entities [4], that internal structure is unknown is then yielded by relation among input and output timed data streams.

This relation can be denoted as :

$$R_k(\langle \alpha_1, a_1 \rangle, \ldots, \langle \alpha_m, a_m \rangle; \langle \beta_1, a_1 \rangle, \ldots, \langle \beta_n, a_n \rangle), \tag{4}$$

where $\langle \alpha_i, a_i \rangle$ is an input timed data stream and $\langle \beta_j, b_j \rangle$ is an output timed data stream, for $0 < i \leq m$ and $0 < j \leq n$.

An interaction is not as trivial as it seems. The black-box program components do not know anything about each other and are not necessarily designed to work with one another. The chance that they have compatible communication periods is very small. This means that *glue code* that connects two components and composes a pair has to implement an interaction protocol, it's relation that somehow compensates the mismatch of their periods.

5. COMPLEX PROGRAM SYSTEM

A *complex program system P* is from the ABT's view defined as a component-based system, which consists of program component instances pk_i and connectors k_i (*glue code*) between them, which are together modelled as *abstract behavior types (ABTs)*.

Distinction between a component's ABT and a connector's ABT is just that a component's ABT is an atomic ABT whose internal structure is unknown, whereas a connector's ABT is known to be an ABT that is itself composed out of other ABTs

Since *glue code* is between components which connects and through which can components communicate with their environment. It implies that :

1. the components must be amenable to external coordination control what implicate to *glue code* must contain suitable mechanism through which components can interact with their environment and
2. the glue code must contain constructs to provide such external coordination what implicates to *glue code* programming language must incorporate a coordination model.

A complex program system P, Fig. 1, can be denoted as :

$$P = \left\{ pk_1 \overset{k_1}{\circ} pk_2 \overset{k_2}{\circ} \ldots \overset{k_{n-1}}{\circ} pk_n \right\}. \tag{5}$$

where pk_i is component's ABT and k_i is connector's ABT , for $0 < i \leq n$.

ISSN 1335-8243 (print) © 2012 FEI TUKE
www.aei.tuke.sk

ISSN 1338-3957 (online)
www.versita.com/aei

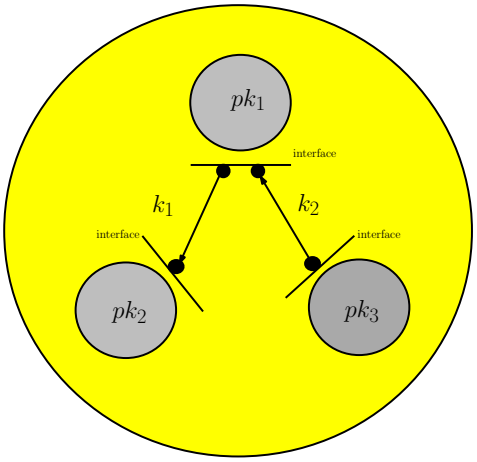

Fig. 1 Complex Program system consists of three program components and two connectors.

5.1. Connectors

Definition 5.1. *A* connector *can be in general defined as a medium of communication with exactly two ends, and a relation that defines its interaction protocol through these ends.*

□

We recognizes two types of connector's end:

- *source end* - through this end timed data stream enter into channel;
- *sink end* - through this end timed data stream come out of the channel;

Connector's simplest types are channels (Fig.:2). Channels as a set of simplest *glue code* was firstly defined in REO by Dr. Fahrad Arbab [1], [2], [3], [8], [9]. There is no requirement in channels to have defined a source and a sink end. It is perfectly content with a channel that has two sources or two sink, but there have to be defined behavior of the channel through relation which defines interaction protocol of the channel.

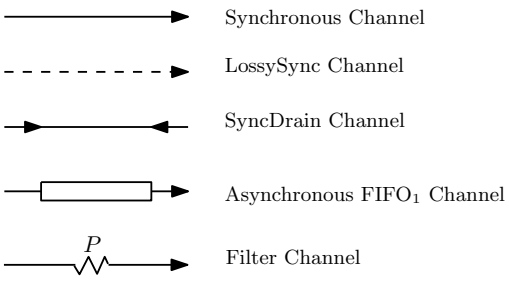

Fig. 2 Some simplest connectors - Channels

Synchronous channel

Synchronous channel is a channel on which ends are read-operations and write-operations succeed synchronously.

Behavior of Sync ABT is defined by the relation

$$\langle \alpha, a \rangle \text{Sync} \langle \beta, b \rangle \equiv \alpha = \beta \wedge a = b. \tag{6}$$

This notation represents the behavior of any entity in which :

- output data stream is identical to its input data stream $(\alpha = \beta)$;
- element on the output is produced at the same time as element on its input is consumed $(a = b)$;

LossySync Channel

LossySync is a synchronous channel with a behavior very similar to Synchronous channel except that it is always ready to consume data items written to its source. Behavior of LossySync ABT is defined by the relation

$$\langle \alpha, a \rangle \text{LossySync} \langle \beta, b \rangle \equiv \begin{cases} \langle \alpha, a \rangle \text{LossySync} \langle \beta, a(0).b' \rangle & \text{for } a(0) > b(0), \\ \beta(0) = \alpha(0) \wedge \langle \alpha', a' \rangle \text{LossySync} \langle \beta', b' \rangle & \text{for } a(0) = b(0), \\ \langle \alpha', a' \rangle \text{LossySync} \langle \beta, b \rangle & \text{otherwise.} \end{cases} \tag{7}$$

This notation represents the behavior of any entity that :

- all write-operations on the source end of a LossySync channel are immediately succeed;
- if there is a pending take on its sink end, then the written data item is transferred;
- otherwise, the write-operation succeeds, but the written data item is lost;

SyncDrain Channel

SyncDrain is a synchronous channel and the behavior of SyncDrain ABT is defined by the relation

$$\langle \alpha, a \rangle \text{SyncDrain} \langle \beta, b \rangle \equiv a = b. \tag{8}$$

This notation represents the behavior of any entity that :

- produces no output data stream, because it has no sink end;
- every data item written to its source end is lost;
- write-operation on the one of the end remains pending until write on other end is performed, in other world operations on both ends are performed at the same time $(a = b)$ and write operations can be blocked if it is necessary, to ensure that they succeed atomically;

ISSN 1335-8243 (print) © 2012 FEI TUKE
www.aei.tuke.sk

ISSN 1338-3957 (online)
www.versita.com/aei

Asynchronous channel

The behavior of an asynchronous channel ABT FIFO_k with the bounded capacity of k, for $k > 0$ is defined by the relation

$$\langle \alpha, a\rangle \ \text{FIFO}_k \ \langle \beta, b\rangle \quad \equiv \quad \alpha = \beta \wedge a < b < a^{(k)}, \quad \text{where} \quad b(i) < a(i+k). \tag{9}$$

The FIFO_k ABT represents the behavior of any entity in which :

- output data stream is identical to its input data stream $(\alpha = \beta)$;
- every element on its output some time after its respective input element is observed $(a < b)$ but before its k^{th}-next input element is observed $(b < a(k)$ which means $b(i) < a(i+k)$, for $i > 0)$;

Observe that FIFO_1 is indeed a special case of FIFO_k with k = 1. FIFO_1 is an asynchronous channel with :

1. a source end,
2. a sink end and
3. bounded buffer with the capacity of one data item.

Its buffer is initially empty. A write operation on its source end succeeds and fills the buffer. If the buffer is non-empty, a write operation is succeeded on the sink end and removed from buffer. Other input resp. output operations are block and waiting for change the status of the buffer.

Filter channel

FILTER(P) channel behaves like a Sync channel, except that only those data items that match the pattern P can actually pass through it; others are always accepted by its source, but are immediately lost. The behavior of FILTER ABT is defined by the relation

$$\begin{aligned} &\langle \alpha, a\rangle \text{FILTER}(P) \langle \beta, b\rangle \equiv \\ &\equiv \begin{cases} \beta(0) = \alpha(0) \wedge b(0) = \\ \quad = a(0) \wedge \langle \alpha', a'\rangle \text{FILTER}(P) \langle \beta', b'\rangle & \text{if } \alpha(0) \ni P, \\ \langle \alpha', a'\rangle \text{FILTER}(P) \langle \beta, b\rangle & \text{otherwise.} \end{cases} \end{aligned} \tag{10}$$

The infix operator $\alpha(0) \ni P$ denotes whether the data item $\alpha(0)$ matches with the pattern P or not and therefore if $\alpha(0)$ passes through or not. When $\alpha(0)$ does not pass through channel, it is lost, and the ABT proceeds with the rest of its timed data streams.

5.2. Category of program components

An interface of program component INF_{pk_s} is a set of all program component contact ports $cp_{pk_{s_r}}$ and can be denoted as :

$$INF_{pk_s} = \{cp_{pk_{s_1}}, cp_{pk_{s_2}}, \ldots, cp_{pk_{s_r}}\} \tag{11}$$

while all program components contact ports $cp_{pk_{s_r}}$ and relation among observable input and output timed data streams $R_k(\langle \alpha_i, a_i\rangle; \langle \beta_j, b_j\rangle)$ forms the category $\mathscr{P}$komp, category of program components $\mathscr{P}$komp.

Since program components pk_s communicates with their environment through the contacts ports sending untargeted passive messages, which only supports the operations of writing and reading, we can also say that each program component contact port $cp_{pk_{s_r}}$ is a type :

$$cp_{pk_{s_r}} = \langle \alpha(i) : \text{Stream}[T], a(i) : \text{E}\rangle \tag{12}$$

where Stream[T] is a type of the data item $\alpha(i)$ and E is a type of a time moment $a(i)$, for $0 < i \leq n$.

The category $\mathscr{P}$komp is a category, where :

- objects are individual component interfaces INF_{pk_s} which are products of the contact ports $cp_{pk_{s_r}}$:

$$INF_{pk_s} = \{cp_{pk_{s_1}} \times \ldots \times cp_{pk_{s_r}}\} \tag{13}$$

- morphisms are interactions among observable input timed data stream and output timed data stream in the form :

$$\langle \alpha_i, a_i\rangle \xrightarrow{\text{interaction}_k} \langle \beta_j, b_j\rangle,$$

which is yielded by relations and can be denoted in the form :

$$R_k(\langle \alpha_i, a_i\rangle; \langle \beta_j, b_j\rangle). \tag{14}$$

5.3. Behavior of complex program system

Behavior of complex program system is expressed as conjunction of maximal relations among a set of timed data streams. In other words, complex program system behavior $B(P)$ is expressed as union of individual ABTs, because abstract behavior type (ABT) is a maximal relation among a set of timed data streams what can be denoted as :

$$B(P) = \{ABT_{pk_1} \cup ABT_{pk_2} \cup \ldots ABT_{pk_s}\} \tag{15}$$

or

$$B(P) = \begin{array}{ll} \{R_{max}(\langle \alpha_{pk_1}, a_{pk_1}\rangle; \langle \beta_{pk_1}, b_{pk_1}\rangle) & \cup \\ \cup \quad R_{max}(\langle \alpha_{pk_s}, a_{pk_s}\rangle; \langle \beta_{pk_s}, b_{pk_s}\rangle)\} & \end{array} \tag{16}$$

6. CONCLUSIONS

Complexity of the component-based program systems are constantly growing up and are more and more used. But if we want to reach expected results, we must be able to model their behavior. Behavior modelling is therefore very important.

Complex program system from ABT view is composed from instances of components and connectors, *glue code* between them which provide interaction, communication among components and both are modelled as abstract behavioral types (ABTs). Program components ABT is an atomic ABT whose internal structure is unknown, whereas a connectors ABT structure is known and connectors ABT can be itself composed of other connectors ABTs to more complex ABTs.

ABT as model for program component in contrast to classical abstract data type (ADT) represents higher level of the abstraction as ADT and it's inter-component communication protocol is based on sending untargeted passive messages through component contact ports which only read and write operations are supported. Sending message is just *write-operation* and receiving is just *read-operation* and set of all component contact ports represents interface of the program component.

Interaction between program components, which are independent entities that internal structure is unknown is not as trivial as it in first seems. The black-box program components do not know anything about each other and are not necessarily designed to work with one another and that means that connector that connects two components and composes pair has to implement an interaction protocol which is yielded by relation among input and output timed data stream

$$R_k(\langle\alpha_1, a_1\rangle, \ldots, \langle\alpha_m, a_m\rangle; \langle\beta_1, a_1\rangle, \ldots, \langle\beta_n, a_n\rangle).$$

In this paper we have defined relational category of program components. Objects of this relational category are individual component interfaces which are products of the contact ports and morphisms are interactions among observable input timed data stream and output timed data stream which is yielded by relations. Category of program components will be used in our research [10], [11], [12], [13] as model for behavioral description of the component-based complex program systems.

ACKNOWLEDGEMENT

This work is the result of the project implementation: Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES

[1] ARBAB, F.: *Composition by Interaction.* Inaugural Lecture (2005). Leiden Institute of Advanced Computer Science, Faculty of Mathematics & Natural Sciences, Leiden University.

[2] ARBAB, F.: *Abstract Behavior Types: A foundation model for components and their composition.* Science of Computer Programing 55, 2005, pp. 3–52.

[3] ARBAB, F.: *REO: a channel-based coordination model for component compostition.* Mathematical Structures in Computer Science 14, 2004, pp. 329–366.

[4] KRISTENSEN, B. – MAY, D.: *Component Composition and Interaction.* In Proceedings of International Conference on Technology of Object-Oriented Languages and Systems, 1996, TOOLS PACIFIC'96.

[5] CHEN, X. – JIFENG, H. – LIU, Z. – ZHAN, N.: *A Model of Component-Based Programming.* Report No. 350, 2006, UNU-IIST, P.O.Box 3058, Macao.

[6] SZYPERSKI, C.: *Component Software : Beyond Object-Oriented Programming* Addison-Wesley, 1997.

[7] BAIER, Ch. – SIRJANI, M. – ARBAB, F. – RUTTEN, J.: *Modeling Component Connectors in Reo by Constraint Automata* Science of Computer Programming 61, 2006, pp. 75–113.

[8] CLARKE, D. – PROENCA, J. – ARBAB, F. – LAZOVIK, A.: *Deconstruction REO* Electronic Notes in Theoretical Computer Science, Volume 229 (2), 2009, pp. 43–58.

[9] KOKASH, N. – ARBAB, F.: *Applying REO to service coordination in long-running business transaction*, SAC'09, 2009, pp. 1381–1382.

[10] SLODIČÁK, V.: *Some useful structures for categorical approach for program behavior*, Journal of Information and Organizational Sciences, Vol. 35, No. 1, 2011, pp. 93–103, www.jos.foi.hr.

[11] MIHÁLYI, D. – NOVITZKÁ, V.: *A Coalgebra as an Intrusion Detection System*, Acta Polytechnica Hungarica, Budapest, Vol. 7, Issue 2, (2010), pp. 71–79.

[12] JENČIK, M. – MIHÁLYI, D.: *Formal description of behaviour for large program system*, In CSE 2010: proceedings of International Scientific conference on Computer Science and Engineering, Košice-Stará Ľubovňa, Slovakia, (2010), pp. 15–22.

[13] JENČIK, M. – NOVITZKÁ, V.: *Formal Description of Behaviour for Object-based Programs*, In Electrical Engineering and Informatics 2: Proceeding of Faculty of Electrical Engineering and Informatics of the Technical University of Košice, (2011), pp. 223–228.

Received January 12, 2012, accepted April 16, 2012

BIOGRAPHIES

Marián Jenčik graduated (MSc) at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice in 2004. Since 2008 he is studying as external PhD. student

at original department. His scientific research is focusing on modelling software object in category theory and defing suitable categorical structure for coalgebraic behavior description of object-oriented and component-based program systems.

Daniel Mihályi worked as a researcher at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice and later as a assistant professor. In 2009 he defended PhD. thesis Duality of formal description of construction and program behavior. The main area of his research includes application of category theory in informatics and using source-based logical systems for formal description of program systems behavior.

ISSN 1335-8243 (print) © 2012 FEI TUKE
www.aei.tuke.sk

ISSN 1338-3957 (online)
www.versita.com/aei