Communication with Environment in Alvis Models
Marcin Szpyrka, Piotr Matyasik, Rafał Mrówka, and Leszek Kotulski

Abstract—Alvis is a modelling language defined for the design and a formal verification of embedded systems. An Alvis model is a system of agents that usually run concurrently, communicate with another, compete for shared resources etc. Due to the fact that an embedded system usually collects inputs that come from its environment and provides outputs that go to the environment it is necessary to provide a mechanism to describe such a communication. In contrast to another formal languages used to model embedded systems it is not necessary, using Alvis, to design such an environment as a part of a model. The paper deals with the problem of modelling a communication with an embedded system environment with Alvis.

Keywords—Alvis modelling language, embedded systems, formal verification, communication with environment.

I. INTRODUCTION

Alvis\textsuperscript{1} [1], [2] is a novel modelling language designed for real-time systems, especially for embedded ones. The main goal of the Alvis project was to strike a happy medium between formal and practical modelling languages. Formal methods like Petri nets [3], [4], [5], time automata [6], process algebra [7] etc. are very seldom used in real IT projects due to their specific mathematical syntax. The Alvis syntax seems to be more user-friendly. From programmers point of view, it is necessary to design two layers of an Alvis model. The code layer uses Alvis statements supported by the Haskell functional programming language to define a behaviour of individual agents. The graphical layer (communication diagram) is used to define communication channels among agents. The layer takes the form of a hierarchical graph that allows designers to combine sets of agents into modules that are also represented as agents (called hierarchical ones). The Alvis language is supported by Alvis Toolkit software that, among other things, provides Alvis Editor used for developing Alvis models and Alvis Translator used to generate LTS graphs (Labelled Transition System). Such an LTS graph is a formal representation of an Alvis model can be formally verified e.g. with the help of the CADP toolbox [8].

An embedded system is one that is a part of a larger one. It is surrounded by other parts of the larger system that constitute the embedded system environment. Such an embedded system collects inputs that come from its environment (from sensors) and provide outputs that go to the environment (to controllers). To verify an embedded system formally we cannot separate it from its environment. Thus, if a formal language is used e.g. Petri nets, time automata, process algebra etc., an embedded system model must include both the system and its environment. As a result of such a situation a model is often significantly more complex and the state explosion problem makes a formal verification difficult or even impossible. One of the main Alvis advantages is the possibility of a flexible specification of a behaviour of an embedded system’s environment. Instead of describing an environment as a part of an Alvis model, it is possible to specify signals/values sent or received by the environment.

The paper focuses at the modelling of a communication with an embedded system environment in Alvis models. It is organised as follows. Section II provides a short presentation of the Alvis modelling language. Border ports used for a communication with an embedded system environment are described in Section III. A formal description of Alvis models is given in Section IV. Section V deals with methods of communication with environment in Alvis. The paper is summarised in the final section.

II. ALVIS LANGUAGE

Alvis is a successor of the XCCS modelling language [9], [10], which was an extension of the CCS process algebra [11], [7]. An Alvis model is composed of three layers:

• **Graphical layer** — is used to define data and control flow among distinguished parts of the system under consideration that are called agents. The layer takes the form of a hierarchical graph and supports both top-down and bottom-up approaches to systems development. Graphical items used in a communication diagram design are shown in Fig. 1.
• **Code layer** — is used to describe the behaviour of individual agents. It uses both Haskell functional programming language [12] and original Alvis statements [1], [2]. The set of Alvis statements is given in Table I.
• **System layer** — depends on the model running environment i.e. the hardware and/or operating system. The layer is the predefined one. The system layer is used for a simulation and verification purposes.

Agents in Alvis are divided into three groups: active, passive and hierarchical agents. Active agents (see Fig. 1 agent $A$) perform some activities and are similar to tasks in the Ada programming language [13]. Each of them can be treated as a thread of control in a concurrent system. Passive agents (agent $B$) do not perform any individual activity, and are similar to protected objects (shared variables). Hierarchical agents (agent $C$) represent submodels (modules).

An agent can communicate with other agents through ports. Ports are drawn as circles placed at the edges of the corresponding rounded box or rectangle. A communication channel is defined explicitly between two agents and connects two ports. Communication channels are drawn as lines. An arrowhead points out the input port for the particular connection.

\footnotesize
\item M. Szpyrka, P. Matyasik, R. Mrówka, and L. Kotulski are with AGH University of Science and Technology, Department of Applied Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland (e-mails: {mszpyrka; ptm; Rafał.Mrowka; kotulski}@agh.edu.pl).
\item Project web site: [http://fm.ia.agh.edu.pl](http://fm.ia.agh.edu.pl)

---

\textsuperscript{1}
Selects one of the alternative choices. 

Collects a signal via the port 

Empty statement. 

Starts the agent 

Turns on border ports. 

Define a set of statements that must be 

Repeats execution of the loop contents while 

Transfers the control to the line of code 

Delays an agent execution for a given number 

of milliseconds. 

Evaluates the expression and assigns the result 

to the parameter; the exec keyword can be omitted. 

Terminates an agent or a procedure.

Conditional statement. 

Sends a value of the parameter 

A border port is used for a parameterless communication, then 

so it must be described both with the 
in

or

out

clause. 

If a border port is used both as an input and output one,

Then it must be described both with the

in

and

out

clauses. 

It should be underlined that only the

signal

modifier should be used in the final model of an embedded system. Other modifiers are defined mainly for the verification purposes, if reduced models are considered.

Let us focus on the description of input border ports presented in Fig. 2. Signals’ directions are considered from an embedded system point of view, thus all considered ports are used to send information from an environment to the

(connection 

(A,p3,B,q)). Communication channels without arrowheads represent connections with opposite directions (connection between ports 

A,p2 and 

C,r).

The code layer is used to define data types used in the model under consideration, functions for data manipulation and the behaviour of individual agents. The layer uses the Haskell functional language (e.g. the Haskell type system) and original Alvis statements. The set of Alvis statements is given in Table I. To simplify the syntax, the following symbols have been used. 

A stands for an agent name, 

p stands for a port name, 

x stands for a parameter, 

g, g1, g2, ... stand for guards (Boolean conditions), 

e stands for an expression and 

ms stands for milliseconds.

Each non-hierarchical agent placed in a communication diagram must be defined in the corresponding code layer and vice-versa. Besides the statements presented in Table I, Alvis provides the environment statement that can be used only in a code layer preamble. This statement is used to specify behaviour of border ports (see Section III).

The system layer considered in the paper is denoted by 

α1. The 

α1 system layer has been worked out to make Alvis suitable for the modelling of single-processor embedded systems. Frankly speaking, 

α1 stands for a set of system layers that differ about the scheduling algorithm. The 

α1 layers are based on the following assumptions: 

• All active agents share the same processor.

• The predefined 

α1 scheduler function is called after each statement automatically and makes agents running as soon as possible.

III. BORDER PORTS

Border ports can be used both for collecting or sending some information to the embedded system environment. They cannot be connected with any other ports. Properties of border ports are specified in the code layer preamble with the use of the environment statement. It is possible to specify (in the code layer) all details of signals that a system collects or sends through any border port. Such ports must have unique names. The same name of a border port used twice means that two agents use the same border port.

Each border port used as an input one is described with at least one 
in

clause. Similarly, each border port used as an output one is described with at least one 
out

clause. Each clause inside the environment statement contains the following pieces of information [14]: 

• 
in

or 

out

key word,

• the border port name,

• a type name or a list of permissible values to be sent through the port,

• a list of time points, when the port is accessible,

• optionally some modifiers: 

durable,

queue,

signal.

If a border port is used both as an input and output one, then it must be described both with the 
in

and 

out

clauses. If different kinds of signals can be sent through a border port, then more than one 
in

or 

out

clause must be used. If a border port is used for a parameterless communication, then the first list is empty. Similarly, if a border port is always accessible, then the second list is empty. Lists are defined using the Haskell language. In particular, it is possible to use infinite lists [12].

It should be underlined that only the 

signal

modifier should be used in the final model of an embedded system. Other modifiers are defined mainly for the verification purposes, if reduced models are considered.

Let us focus on the description of input border ports presented in Fig. 2. Signals’ directions are considered from an embedded system point of view, thus all considered ports are used to send information from an environment to the

Fig. 1. Elements of Alvis communication diagrams.

TABLE I

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>Empty statement</td>
</tr>
<tr>
<td>out p x</td>
<td>Sends a signal via the port p.</td>
</tr>
<tr>
<td>out x p</td>
<td>Sends a value via the port p and assigns it to the parameter x.</td>
</tr>
<tr>
<td>jump label</td>
<td>Transfers the control to the line of code identified with the label.</td>
</tr>
<tr>
<td>jump far A</td>
<td>Transfers the control to the agent A.</td>
</tr>
<tr>
<td>loop (g) [...]</td>
<td>Repeats execution of the loop contents while the guard if satisfied.</td>
</tr>
<tr>
<td>loop (every ms) [...]</td>
<td>Repeats execution of the loop contents every ms milliseconds.</td>
</tr>
<tr>
<td>loop [...]</td>
<td>Infinite loop.</td>
</tr>
<tr>
<td>exec x = e</td>
<td>Evaluates the expression and assigns the result to the parameter; the exec keyword can be omitted.</td>
</tr>
<tr>
<td>exit</td>
<td>Terminates an agent or a procedure.</td>
</tr>
<tr>
<td>if (g1) [...] elseif (g2) [...] else [...]</td>
<td>Conditional statement.</td>
</tr>
<tr>
<td>in p x</td>
<td>Collects a signal via the port p.</td>
</tr>
<tr>
<td>in p</td>
<td>Collects a value via the port p and assigns it to the parameter x.</td>
</tr>
<tr>
<td>p</td>
<td>A border port used both as an input and output one.</td>
</tr>
<tr>
<td>alt [...]</td>
<td>Selects one of the alternative choices.</td>
</tr>
<tr>
<td>start A</td>
<td>Starts the agent A if it is in the init state, otherwise does nothing.</td>
</tr>
<tr>
<td>sti</td>
<td>Turns on border ports.</td>
</tr>
</tbody>
</table>
COMMUNICATION WITH ENVIRONMENT IN ALVIS MODELS

in a [1..4] []; 
in b [1..4] (map (100+) [1..]); 
in c [1..4] (map (100+) [1..]) signal; 
in d [1..4] (map (100+) [1..]) durable; 
in e [1..4] (map (100+) [1..]) queue; 
in f [1..4] (map (100+) [1..]) signal durable; 
in g [1..4] (map (100+) [1..]) signal queue;

Fig. 2. Examples of input border ports’ specification.

c The port behaves similarly to the b one, but the accessibility of the port is not guaranteed.
d Any of the values 1, 2, 3, 4 can be sent through the port every 100 time-units, but if the system is not ready to send a value then the environment waits for it.
e The port behaves similarly to the c one, but the opportunities are put into a FIFO queue.
f The port behaves similarly to the d one, but the accessibility of the port is not guaranteed.
g The port behaves similarly to the e one, but the accessibility of the port is not guaranteed.

Fig. 3. Examples of output border ports’ specification.

Let us focus on the description of output border ports presented in Listing 3.
a Any of the values 1, 2, 3, 4 can be sent through the port at any time point. An agent that performs the out statement sends the value immediately (never waits for the port accessibility).
b Any of the values 1, 2, 3, 4 can be sent through the port every 100 time-units, but if the system is not ready to send a value then the opportunity is lost.

c The port behaves similarly to the b one, but the accessibility of the port is not guaranteed.

A hierarchical communication diagram can be transformed into a non-hierarchical one with the analysis operation [15]. Thus, from the formal description point of view it is necessary to consider models with non-hierarchical communication diagrams only.

Let \( P(X) \) denote the set of ports of an agent \( X \). We can distinguish the following subsets of the set \( P(X) \):

- \( P_{\text{border}}(X) \) denotes the set of border ports of agent \( X \) i.e. ports that are specified in the environment statement.
- \( P_{\text{internal}}(X) = P(X) - P_{\text{border}}(X) \) denotes the set of internal ports of agent \( X \).
- \( P_{\text{in}}(X) \) denotes the set of input ports of agent \( X \). An input border port is a border port with at least one in specification. An input internal port is an internal port with at least one one-way connection leading to this port or with at least one two-way connection.
- \( P_{\text{out}}(X) \) denotes the set of output ports of agent \( X \). An output border port is a border port with at least one out specification. An output internal port is an internal port with at least one one-way connection leading from this port or with at least one two-way connection.
- \( P_{\text{unc}}(X) = P(X) - (P_{\text{in}}(X) \cup P_{\text{out}}(X)) \) denotes the set of unconnected ports.
- \( P_{\text{proc}}(X) \subseteq P_{\text{internal}}(X) \) denotes the set of procedure ports of agent \( X \) (for passive agents only) i.e. ports with defined the proc statement (names of such ports are treated as names of procedures).

For a set of agents \( W \) we define sets: \( P(W) = \sum_{X \in W} P(X) \), \( P_{\text{in}}(W) = \sum_{X \in W} P_{\text{in}}(X) \), etc. Moreover, let \( P \) denote the set of all model ports, \( P_{\text{in}} \) denote the set of all model input ports, etc.

**Definition 1:** A Non-hierarchical communication diagram is a triple \( D = (A, C, \sigma) \), where:

- \( A = \{X_1, \ldots, X_n\} \) is the set of agents consisting of two disjoint sets, \( A_A, A_P \) such that \( A = A_A \cup A_P \), containing active and passive agents respectively.
\( C \subseteq \mathcal{P} \times \mathcal{P} \) is the communication relation, such that:
\[
\forall X \in \mathcal{A}: (\mathcal{P}(X) \times \mathcal{P}(X)) \cap C = \emptyset, \tag{1}
\]
\[
((\mathcal{P}_{\text{border}} \times \mathcal{P}) \cup (\mathcal{P} \times \mathcal{P}_{\text{border}})) \cap C = \emptyset, \tag{2}
\]
\[
\mathcal{P}_{\text{proc}} \cap \mathcal{P}_{\text{in}} \cap \mathcal{P}_{\text{out}} = 0, \tag{3}
\]
\[
(p,q) \in (\mathcal{P}(\mathcal{A}_{\text{A}}) \times \mathcal{P}(\mathcal{A}_{\text{P}})) \cap C \Rightarrow q \in \mathcal{P}_{\text{proc}}, \tag{4}
\]
\[
(p,q) \in (\mathcal{P}(\mathcal{A}_{\text{P}}) \times \mathcal{P}(\mathcal{A}_{\text{A}})) \cap C \Rightarrow p \in \mathcal{P}_{\text{proc}}, \tag{5}
\]
\[
(p,q) \in (\mathcal{P}(\mathcal{A}_{\text{P}}) \times \mathcal{P}(\mathcal{A}_{\text{P}})) \cap C \Rightarrow \Rightarrow (p \in \mathcal{P}_{\text{proc}} \land q \notin \mathcal{P}_{\text{proc}}) \lor
\]
\[
\lor (q \in \mathcal{P}_{\text{proc}} \land p \notin \mathcal{P}_{\text{proc}}). \tag{6}
\]

\( \sigma : \mathcal{A}_{\text{A}} \rightarrow \{ \text{False, True} \} \) is the start function that points out initially activated agents.

Each element belonging to \( C \) is called a connection or a communication channel. The restrictions from Definition 1 have the following meaning. (1) – A connection cannot be defined between ports of the same agent. (2) – Border ports cannot be connected with any other ports. (3) – Procedure ports are either input or output ones. (4), (5) – A connection between an active and a passive agent must be a procedure call. From conditions (3)–(5) it follows that any connection with a passive agent must be a one-way connection. (6) – A connection between two passive agents must be a procedure call from a non-procedure port.

The start function \( \sigma \) makes possible delaying activation of some agents. Names of agents that are initially activated are underlined in a communication diagram.

Definition 2: A flat Alvis model is a tuple \( \overline{\mathcal{A}} = (D, B, \varphi) \), where:

- \( D = (\mathcal{A}, \mathcal{C}, \sigma) \) is a non-hierarchical communication diagram,
- \( B \) is a syntactically correct code layer,
- \( \varphi \) is a system layer.

Moreover, each agent \( X \) belonging to the diagram \( D \) must be defined in the code layer, and each agent defined in the code layer must belong to the diagram.

It should be underlined that a syntactically correct code layer means also that only input ports may be used as arguments of \( \text{in} \) statements, and only output ports may be used as arguments of \( \text{out} \) statements. From now on, we will consider only \( \overline{\mathcal{A}} = (D, B, \alpha^1) \) models.

Definition 3: A state of an agent \( X \) is a tuple
\[
S(X) = (\text{am}(X), \text{pc}(X), \text{ci}(X), \text{pv}(X)), \tag{7}
\]
where \( \text{am}(X), \text{pc}(X), \text{ci}(X) \) and \( \text{pv}(X) \) denote mode, program counter, context information list and parameters values of the agent \( X \) respectively.

All possible modes and transitions among them are shown in Fig. 4. \text{Finished} means that an agent has finished its work. \text{Init} is the default mode for agents that are inactive in the initial state. An agent can be activated by another one with the \text{start} statement. \text{Running} means that an agent is performing one of its statements. \text{Ready} means that an agent is ready to execute the next statement, but it waits for an access the processor. \text{Taken} means that one of the passive agent procedures has been called and the agent is executing it. For passive agents, \text{waiting} means that the corresponding agent is inactive and waits for another agent to call one of its accessible procedures. For active agents, the mode means that the corresponding agent is waiting either for a communication with another active agent, or for a currently inaccessible procedure of a passive agent.

The program counter points out the current statement of an agent i.e. the next statement to be executed or the statement that has been executed by an agent but needs a feedback from another agent to be completed (e.g. a communication between two active agents). Relationships between the mode and the program counter of an agent are shown in Table II.

The context information list contains additional information about the current state of an agent e.g. if an agent is the \text{waiting} mode, \( \text{ci} \) contains information about events the agent is waiting for. Possible entries put into \( \text{ci} \) lists are given in Table III. If an agent is in the \text{init} or \text{finished} mode, its context information list is empty.

If an Alvis model uses border ports then an extra agent denoted by \( \ast \) and representing the system environment is considered. The state of the environment is described using its \( \text{ci} \) list only. The list contains information about time moments of the border ports accessibility.

Definition 4: A state of a model \( \overline{\mathcal{A}} = (D, B, \alpha^1) \), where \( D = (\mathcal{A}, \mathcal{C}, \sigma) \) and \( \mathcal{A} = \{ X_1, \ldots, X_n \} \) is a tuple
\[
S = (S(X_1), \ldots, S(X_n), S(\ast)). \tag{8}
\]
The \( S(\ast) \) is omitted, if the considered model does not contain border ports.

Let us focus on the \( \text{ci}(\ast) \) list. For each \( \text{in} \) and \( \text{out} \) clause from the \text{environment} statement, the list contains an entry with information about the next port accessibility with respect to

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\text{mode} & \text{pc}(X) \\
\hline
\text{finished} & 0 \\
\text{init} & 0 \\
\text{ready} & \text{current statement} \\
\text{running} & \text{current statement} \\
\text{taken} & \text{current statement of the called procedure} \\
\text{waiting (active agent)} & \text{current statement} \\
\text{waiting (passive agent)} & 0 \\
\hline
\end{tabular}
\caption{Relationships between the Mode and Program Counter}
\end{table}
the clause. There are two exceptions, when such an entry is missing:

1) There is no the next time point when the corresponding port is accessible.

2) The time points list for the entry is empty – the corresponding port is always accessible with respect to the clause.

Suppose, the environment statement contains the following specification for port a:

\[ \text{in a} \{1, 3, 6\} \]

Thus, in the initial state, \( ci(*) \) contains the following entry \( in(a, 1) \) i.e. a signal will be accessible via port a in one time-unit (one millisecond by default). One time-unit later, this entry will be replaced with \( in(a, 2) \). Then, 1 time unit later the entry will be replaced with \( in(a, 1) \), and next with \( in(a, 3) \). After 6 time-units from the beginning, \( ci(*) \) will not contain any entry assigned to the considered clause.

If a border port is specified with more than one in clause then the clauses distinguished by their indices. Let a border port a be specified as follows:

\[ \text{in a} \{1, 3, 6\} \quad \text{or} \quad \text{in a Bool} \{2, 4, 7, 8, 9\} \]

Thus, in the initial state, \( ci(*) \) contains two entries \( in(a[1, 1]) \), \( in(a[2, 2]) \). One time-unit later, these entries will be replaced with \( in(a[1, 2]), in(a[2, 1]) \), etc. Output border ports are represented in \( ci(*) \) in similar way.

To determine the current mode of an agent that is ready to execute its next statement, the scheduler function \( \alpha^1 \) is used. For any state \( S \), the function is applied to a set of active agents \( A'_A \) and satisfies the following requirements:

- \( \forall X \in A'_A, \alpha^1_2(X) \in \{\text{ready, running}\} \)
- if \( \text{critical} \in ci(X) \) then \( \alpha^1_2(X) = \text{running} \).
- \( |\{X \in A'_A : \alpha^1_2(X) = \text{running}\}| \leq 1 \).

\[ \text{Definition 5: The initial state of a model } \chi = (D, B, \alpha^1) \text{ is a tuple } S_0 \text{ as given in (8), where:} \]

- \( am(X) = \alpha^1_1 S_0(X) \) for any active agent \( X \) such that \( \sigma(X) = \text{True} \);
- \( am(X) = \text{init} \) for any active agent \( X \) such that \( \sigma(X) = \text{False} \);
- \( am(X) = \text{waiting} \) for any passive agent \( X \);
- \( pc(X) = 1 \) for any active agent \( X \) in the ready or running mode and \( pc(X) = 0 \) for other agents.
- \( ci(X) = [ ] \) for any active agent \( X \);
- For any passive agent \( X, ci(X) \) contains names of all accessible ports of \( X \), i.e. names of all accessible procedures) together with the direction of parameters transfer, e.g. \( in(a), out(b) \), etc.
- For any agent \( X, pv(X) \) contains \( X \) parameters with their initial values.

<table>
<thead>
<tr>
<th>Agent ( X )</th>
<th>( am(X) )</th>
<th>( ci(X) ) entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>ready/waiting</td>
<td>( \text{proc}(Y, b, a) )</td>
<td>( X ) has called the ( Y,b ) procedure via port ( a ) and this procedure is being executed in the ( X ) agent context.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{timeout}(s) )</td>
<td>A timer signal for the statement number ( s ) has been generated and waits for serving</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{timer}(s, n) )</td>
<td>A timer signal for the statement number ( s ) will be generated in ( n ) time-units</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{off} )</td>
<td>Border ports handling turned off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( in(s, a) )</td>
<td>A signal generated by the system environment has been provided via border port ( a )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( in(s, a(v)) )</td>
<td>A value ( v ) generated by the system environment has been provided via border port ( a )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( out(s, a) )</td>
<td>The system environment is ready to collect a signal via border port ( a )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( out(s, a(i)) )</td>
<td>The system environment is ready to collect via port ( a ) a value defined by ( i )-th ( out ) clause for the port.</td>
</tr>
<tr>
<td>passive</td>
<td>taken</td>
<td>( \text{proc}(Y, b, a) )</td>
<td>( X ) has called the ( Y,b ) procedure via port ( a ) and this procedure is being executed in the same context as the ( X ) procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>guard</td>
<td>( X ) waits for an open branch of a select statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{critical} )</td>
<td>( X ) is inside a critical section</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{timeout}(s) )</td>
<td>A timer signal for the statement number ( s ) has been generated and waits for serving</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{timer}(s, n) )</td>
<td>A timer signal for the statement number ( s ) will be generated in ( n ) time-units</td>
</tr>
<tr>
<td>passive</td>
<td>waiting</td>
<td>( in(a) )</td>
<td>Input procedure ( X.a ) is accessible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( out(a) )</td>
<td>Output procedure ( X.a ) is accessible</td>
</tr>
<tr>
<td>passive</td>
<td>taken/waiting</td>
<td>( \text{off} )</td>
<td>Border ports handling turned off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( in(s, a) )</td>
<td>A signal generated by the system environment has been provided via border port ( a )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( in(s, a(v)) )</td>
<td>A value ( v ) generated by the system environment has been provided via border port ( a )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( out(s, a) )</td>
<td>The system environment is ready to collect a signal via border port ( a )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( out(s, a(T)) )</td>
<td>The system environment is ready to collect a value of type ( T ) via border port ( a )</td>
</tr>
</tbody>
</table>

\[ * \]

\[ \text{in}(a, n) \]

\[ \text{out}(a, n) \]

\[ \text{out}(a, n) \]
• For any \texttt{in} and \texttt{out} clause from the \texttt{environment} statement a suitable entry (if necessary) is put into $ci(*)$.

We consider behaviour of Alvis models at the level of detail of single steps. More statements \texttt{exec, exit, in,} etc. are \texttt{single-step statements}. On the other hand, \texttt{if, loop and select} are \texttt{multi-step statements}. We use recursion to count the number of steps for multi-step statements. For each of these statements, the first step enters the statement interior. Then, we count steps of statements put inside curly brackets. Steps performed by a model are described using the \texttt{transition idea}. The set of all possible transitions for Alvis models considered in the paper is given in Table IV. Transitions with numbers range from 1 to 15 are called \texttt{system transitions} and transitions with numbers range from 16 to 18 are called \texttt{environment transitions}.

![Table IV: Set of Transitions](image)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$te_{it}$ performs \texttt{cli} statement</td>
</tr>
<tr>
<td>2</td>
<td>$te_{exit}$ enters a critical statement</td>
</tr>
<tr>
<td>3</td>
<td>$te_{delay}$ performs a delay statement</td>
</tr>
<tr>
<td>4</td>
<td>$te_{exe}$ performs an evaluation and assignment</td>
</tr>
<tr>
<td>5</td>
<td>$te_{exit}$ terminates an agent or a procedure</td>
</tr>
<tr>
<td>6</td>
<td>$te_{if}$ enters an if statement</td>
</tr>
<tr>
<td>7</td>
<td>$te_{in}$ performs communication (input side)</td>
</tr>
<tr>
<td>8</td>
<td>$te_{jump}$ jumps to a label</td>
</tr>
<tr>
<td>9</td>
<td>$te_{loop}$ enters a while or infinite loop</td>
</tr>
<tr>
<td>10</td>
<td>$te_{loop}$ enters a loop every loop</td>
</tr>
<tr>
<td>11</td>
<td>$te_{null}$ performs an empty statement</td>
</tr>
<tr>
<td>12</td>
<td>$te_{out}$ performs communication (output side)</td>
</tr>
<tr>
<td>13</td>
<td>$te_{select}$ enters a select statement</td>
</tr>
<tr>
<td>14</td>
<td>$te_{start}$ starts an active agent</td>
</tr>
<tr>
<td>15</td>
<td>$te_{sta}$ performs a sin statement</td>
</tr>
<tr>
<td>16</td>
<td>$te_{inst}$ activates an input border port</td>
</tr>
<tr>
<td>17</td>
<td>$te_{out}$ activates an output border port</td>
</tr>
<tr>
<td>18</td>
<td>$te_{time}$ denotes time passage</td>
</tr>
</tbody>
</table>

Executing of each of presented transitions changes the current state of a model. It is out of the scope of the paper to describe each of the transitions in details. We focus on these transitions that are connected with a communication with environment. More details about the transition idea in Alvis models can be found in [15]. However, the paper describes models with $\alpha^0$ system layer i.e. with unlimited number of processors, and a system environment is not taken under consideration.

V. COMMUNICATION WITH ENVIRONMENT

The fact that a transition $t$ is enabled in a state $S$ with respect to an agent $X$ and that a state $S'$ is the result of executing $t$ in $S$ will be denoted by $S \rightarrow t(X) \rightarrow S'$. Sometimes, an extended version of this notation will be used:

- $S \rightarrow t_{start}(X,Y) \rightarrow S'$, where $Y$ is the argument of the corresponding \texttt{start statement};
- $S \rightarrow t_{in}(X,p,T) \rightarrow S'$, $S \rightarrow t_{out}(X,p,T) \rightarrow S'$, where $X,p$ is the port used for the communication and $T$ is the type of sent/collected value. If necessary, the special \texttt{Empty} type will be used to denote a valueless communication.

Let us start with signals that are generated by an embedded system periodically. Suppose an interrupt signal (without any specified value) is provided to the port $Al.clock$ every 10 ms (see Fig. 5).

![Fig. 5. Collecting periodical signals from environment.](image)

The model presented in Fig. 5 contains only one agent that is initially activated. The set $T$ of all model transitions contains 4 elements:

- $t_{loop}$ \texttt{– enters the main loop interior,}
- $t_{in}$ \texttt{– collects a signal from port $clock$,}
- $t_{exec}$ \texttt{– assigns a new value to parameter $n$,}
- $t_{in}$ \texttt{– provides a signal from environment via port $clock$.}

Let $\Delta(X_i,k)$ denote the duration of the $k$-th step execution for the agent $X_i$. Suppose, $\Delta(A_1,k) = 1$ for any system transition in this model.

The first signal generated by the environment appears in the 10th millisecond. Agent $A_1$ enters the loop, executes the \texttt{in} statement and waits for the signal. Finally, it receives the signal, increases its parameter $n$ of 1 and repeats its behaviour. It should be underlined, that the agent must be ready to collect the signal, otherwise the signal is lost. In this example, the time taken by the three steps the agent realises (entering the loop, executing the \texttt{in} statement and executing the \texttt{exec statement}) is significantly less than 10 ms, thus $A_1$ is always ready to handle the signal.

The initial state is defined as follows:

$S_0 = ((\text{running}, 1, [], (0)), [\text{in}(\text{clock}, 10)])$

Next, $S_0 - t_{loop}^{A_1}(A_1) \rightarrow S_1$, where $t_{loop}^{A_1}(A_1)$ denotes that $\Delta(A_1,1) = 1$ (1 is the step number for the \texttt{loop} statement): $S_1 = ((\text{running}, 2, [], (0)), [\text{in}(\text{clock}, 9)])$.

For a pair of states $S, S'$ we say that $S'$ is \texttt{directly reachable} from $S$ iff there exists $t \in T$ such that $S \rightarrow t \rightarrow S'$. We say that $S'$ is \texttt{reachable} from $S$ iff there exists a sequence of states $S_1, \ldots, S_{k+1}$ and a sequence of transitions $t_1, \ldots, t_k \in T$ such that $S = S_{i-1} \rightarrow t_i \rightarrow S_{i-2} \rightarrow \ldots \rightarrow t_k \rightarrow S_{i+1} = S'$. The set of all states that are reachable from the initial state $S_0$ is denoted by $R(S_0)$.

States of an Alvis model and transitions among them are represented using a labelled transition system (LST graph for short). An LST graph is a directed graph $\text{LTS} = (V, E, L)$, such that $V = R(S_0)$, $L = T$, and $E = \{(S, t, S') : S \rightarrow t \rightarrow S' \land S, S' \in R(S_0)\}$. In other words,
an LTS graph presents all reachable states and transitions among them in the form of the directed graph.

```
A1:(running,1,[](),)
  *: [in(clock,10)]
  loop(A1)/1
A1:(running,2,[](),)
  *: [in(clock,9)]
  in(A1.clock)/1
A1:(running,3,[](),)
  *: [in(clock,10)]
  exec(A1)/1
  loop(A1)/1
  in(A1.clock)/1
A1:(waiting,2,[](),)
  *: [in(clock,8)]
  8/in(*.clock)
A1:(waiting,1,[](),)
  *: [in(clock,8)]
  in(A1.clock)/1
```

Fig. 6. LTS for the model from Fig. 5.

The LTS graph for the model from Fig. 5 is presented in Fig. 6. We have omitted the parameter \( n \) value in order to receive a finite graph. Labels in the presented graph are of the form \( t_1/\text{transitions}/t_2 \), where \( t_1 \) stands for the time the system stays in the \emph{old} state and \( t_2 \) stands for the duration of the step. If any of the time values is equal to 0, it is omitted together with the corresponding slash. The \texttt{loop(A1)}/1 means that the agent executes the \texttt{loop} transition and it takes 1 ms.

Let us focus on the state \( S_2 \) (\( n = 0 \)):

\[ S_2 = \{ \{ \text{waiting}, 2, [\text{in(clock)}] \}, (0) \}, [\text{in(clock, 8)}] \} \]

In the state \( S_1 \) agent \( A \) performs an \texttt{in} step to collect a signal from the input border port \texttt{clock}. There is no signal to collect thus, the agent changes its mode to \texttt{waiting} and waits for such a signal (additional entry is included into the agent \texttt{ci} list). Seven milliseconds later such a signal is generated by the environment that is represented by the \texttt{t_{in}} transition. This transition finalises the agent \( A \) communication with its environment. The signal is collected and the agent program counter takes the next value.

Suppose, \texttt{in(clock)} \notin \texttt{ci2(A1)}. Thus, an execution of the \texttt{t_{in}} transition denotes a generation of a signal by the environment, but the signal is not handled by the system.

Suppose, the border port \texttt{clock} is specified as follows:

\texttt{in(clock []) (map (10*) [1..]) durable; end in(clock) \notin ci(A1) when the \texttt{t_{in}} transition is executed. Thus, as a result of the transition execution the \texttt{in(*.clock)} is included into the \texttt{ci} list. It means that a signal provided via port \texttt{clock} waits for handling. When executing the \texttt{t_{in}} transition, agent \( A \) collects the signal and immediately moves to a state with \texttt{pc(A1) = 3}. If the \texttt{queue} modifier is used instead of \texttt{durable}, a context information list of an agent may contain more than one \texttt{in(*.clock)} entry. Each of such entries represents a signal waiting for handling.

Suppose, the border port \texttt{clock} is specified as follows:

\texttt{in clock [] (map (10*) [1..]) signal;}\texttt{end}

This means that the environment only may provide a signal via port \texttt{clock} every 10 ms, but we have no confidence that such a signal will be generated. This modifier can be connected with \texttt{durable} and \texttt{queue} ones.

To specify that a signal via port may be provided at any time, we can use the following clause:

\texttt{out clock [] [] signal;}

In such a case agent \( A \) still counts the collected signals, but this time there is not any regularity with respect to time points the signals appear. Moreover, some signals can be lost, if they come too often.

Let us focus on the \( A_2 \) agent that is a part of an air conditioner controller. The agent reads the current temperature and turns on/off the air conditioner depends on the comparison of the current temperature with a set threshold temperature. An Alvis model of such a system is given in Fig. 7.

Fig. 7. Collecting a sensor value from the environment.

Using the empty list in the \texttt{ac} border port specification means that agent \( A_2 \) can send a Boolean value via this port any time it is necessary.

The LTS graph for the model from Fig. 7 is shown in Fig. 8. To reduce the graph size we have used letters \( L \) and \( G \) instead of values less than or equal to the threshold and greater than the threshold respectively. The \texttt{time} label represents the passage of time. This transition is executed only if there is no
any other active one and at least one agent contains a timer entry on its context information list.

VI. SUMMARY

Alvis communication with environment methods have been presented in this paper, together with a basic language description. The environment specification was also introduced, which is crucial for an effective LTS generation. From designers point of view, the Alvis system description is composed of two parts. The first one is a graphical system view, which shows agents and the way they communicate one with another. The second one is a textual specification of agents and the environment behaviour. Together with a system layer, they provide a possibility to simulate and verify an Alvis model. A formal verification of Alvis models is based on LTS graphs generated by the Alvis Translator [16] and external tools provided by the CADP package [8]. CADP offers a wide set of functionalities, ranging from step-by-step simulation to massively parallel model-checking. This approach provides a possibility to use also new methods of a state space exploration in future.

In spite of process algebra origins, Alvis seems to be more convenient from engineering point of view than process algebras. Moreover, a system behaviour is related to selected execution scenario, which is determined by the hardware that executes a model. Alvis is able to perform an execution similar to process algebra formalisms with unlimited numbers of processing units ($$\alpha$$ system layer), but also can behave as a one or two processors units with selected scheduling functions.

REFERENCES