Formal Process Specification Language with a Refinement Capability Intended for Compositional Development of Information Systems

Serge A. Stupnikov, Leonid A. Kalinichenko
Institute for Problems of Informatics
Russian Academy of Sciences
E-mail: stupnik@mccme.ru, leonidk@synth.ipi.ac.ru

Abstract

Starting with csp2B specification facilities as the core, this paper extends AMN further with additional capabilities: sequential processes, parallel composition on arbitrary level, hiding, nondeterministic choice including general choice, assignment operator, main timing primitives of Timed CSP (delay, timeout and time prefixing). The resulting process specification language with a refinement capability is intended for compositional development of information systems. An approach for mapping of extensions of the core into AMN and algorithms of their conversion into B machines are defined.

1 Introduction

Modern globally spread information technologies (such as Web and CORBA) favour development of information systems (IS) reusing existing data and program components arranged around the interoperating middleware. In frame of the SYNTHESIS project, a new, compositional IS development method [3] is being elaborated for correct composition of existing components semantically interoperable in the context of a specific application. This aproach consists in identification of relevant fragments of the component type specifications and constructing their compositions satisfying specifications of requirements. For compositional development the type specifications and their reducts [3] are chosen as the basic units of specification manipulation. For their manipulation, the calculus of type specifications is introduced [12]. The operations of the calculus are intended for producing the specifications of the respective compositions of their operands.

The *most common reduct* (MCR) for a pair of ontologically relevant types T_1 (type of specification of requirements) and T_2 (type of a component specification) is such maximal reduct of T_1 that there exists a reduct of T_2 that refines this reduct of T_1. A procedure for MCR construction is the basic one for identification of types reusability [12]. The common reducts discovered by the procedure are used in type calculus expressions to define new types that should be constructed at the design phase.

This approach is well applicable to component type specifications where concurrent behaviour is not taken into account. Correctness of the results of the design in this case can be verified using formal facilities of formal languages (e.g., such as Z, Object Z, Abstract Machine Notation [17, 6, 13, 1]) and related tools where the concept of refinement is well supported (as in B-Toolkit [2]).

Alongside with program and information components of IS, the process components are also widely spread. Workflows supported by numerous software products on the market constitute the well known technological example of the kind. Processes implemented as workflows are important components of IS. OMG and WfMC [19, 18] efforts allowed to unify various workflow specification languages making possible to consider workflows as components for their interoperable reuse in IS.

In [10, 11] workflow compositional development has been attempted for the scripted type (based on a coloured Petri Net model). There the workflow design has been considered as a process of systematic transformations of pre-existing workflow specifications (taking their reducts and compositions) to get a design refining specification of requirements. Due to the difficulties of refinement proof in coloured Petri Net the application of this modeling approach appeared to be quite limited.

This paper attempts development of process specifications intended for compositional IS development applying another class of languages oriented on the process control, such as CSP, Timed CSP, CCS [8, 15, 16, 14]. For compositional development these facilities should be combined with the formal languages formalizing the refinement concept and the technologies providing for proof of refinement relationship between components. Abstract Machine Notation (AMN) and B-technology [1, 2] provide such possibilities. This is why AMN has been considered as a formal model for the SYNTHESIS language intended for description, design and programming of interoperable information resource environments [9]. Creating a combination of a process control language with the refinement capabilities of B-technology would provide for the compositional development of IS with the process components.

Good move in this direction is the csp2B tool providing the process specification by a subset of the CSP-notation operators [4, 5]. csp2B converts the CSP-like description into standard machine-readable B specifications, the process alphabet events are considered as B-machine operations.

A subset of CSP process constructing operators supported by csp2B is quite limited. Only prefixing, choice, conditional operator, deadlocked process STOP , synchronous and asynchronous parallel composition applicable at the outermost level are supported.

The sequential processes, allowing to decompose the task into the subtasks, together with a parallel composition (defined on arbitrary level) providing for decomposition into the parallel subprocesses and further their join into one thread of a process are additionally required. The hiding providing for abstraction from some functionality of the system is important for component-based development where a part of the initial specification may be substituted by a component specification with some hidden functionality.

Nondeterminism intended for abstraction from specific details of implementation is also helpful. A nondeterministic process can be implemented in various ways characterized by specific implementation features. Thus a high level of abstraction of specification can be reached. The hiding operation can not be used without providing of nondeterministic choice.

An interrupt process required for description of exceptions is an important programming paradigm. Communication between processes of the same specification, not only communication with the environment should be allowed.

The objective of this work is, following the csp2B ideas, to extend AMN further by additional, comprehensive capabilities: sequential processes, parallel composition on the arbitrary level, hiding, nondeterministic choice including general choice, assignment operator, main timing primitives of Timed CSP (delay, timeout and time prefixing). Convenient shorthands of CSP (μ −operation and multiple choice) will provide for readability of specifications. For the AMN extension specific mechanisms have been defined to map the process specifications into AMN as well as the algorithms of converting the CSP machine specifications in the extended notation into B-machines. The tool converting the process description into the standard B-machine should be based on these algorithms. Because of the limited size of the paper, only the sequential operator is considered to show mechanisms of its specification mapping into AMN and algorithms of its conversion into AMN.

The paper is structured as follows. In section (2) the core of the process specification language is considered and principles of mapping of its extension into AMN are considered. In section (3) the mechanisms of mapping the CSP machines containing sequential operators into AMN and algorithms of converting the respective specifications into AMN are considered. In section (4) an example of the sequential operator application – a part of a lift system specification and its mapping into AMN is presented. In section (5) an application of the AMN refinement capabilities for the compositional IS development is discussed. In Appendix the general syntax of CSP-machines is introduced. The conclusion summarizes the results obtained.

2 Principles of implementation of the extension of the core process specification language

The process specification language considered and its implementation in AMN will be considered in two steps. We start with presenting of the part of a language that constitutes its core. After that we shall introduce its extension and an approach for its implementation as the main contribution of this paper. Syntax of the whole language defining general form of CSP machines is given in Appendix.

The language core contains those operators of the language that most naturally can be mapped into AMN. Actually, just these operators were introduced and supported by csp2B. The process control operators of the core include prefixing, choice, conditional, deadlocked process STOP , synchronous and asynchronous parallel composition applicable only at the outermost level.

In the manner of channels in CSP, events from the alphabet of the process may be parameterised by input parameters $In?x$ or output parameters $Out!x$ (see examples in [4]). Then translated into B, these parameters will correspond to the input and output parameters of an operation. Input parameters may also be dot parameters, which means that a process is only willing a particular value as an input parameter rather then being willing to accept any value. Process may be indexed by parameters and these index parameters become state variables in the generated B-machine (see *Free* and *Idle* examples from [4]).

A brief overview of a CSP machine mapping to AMN and algorithms of converting CSP machine definitions into AMN (supporting such mapping) for the language core is given here as they have been defined for the csp2B tool [4]. CSP-like descriptions of system behaviour are to be converted into standard machine-readable B specifications. Proof obligations can be generated for the specifications obtained. A CSP description may be used as a refinement of another CSP or B machine. Also a CSP machine may be used to constrain the execution order of operations of a standard B machine.

Now we briefly consider the CSP machine semantics and the process of B-machine generation for a notation of the language core. In such notation, a process is described by a set of equations of the form

$$I_i(v) = P_i,$$

where I_i is a process identifier and P_i is a process term which may contain several process identifiers [4]. A process term P is said to be in *normal form* if either $P =$ STOP or P is of the form

$$P = Q_1 \,[]\, \ldots \,[]\, Q_n,$$

where each Q_i is of the form

$$\text{IF } G \text{ THEN } a \to I,$$

and I is a process identifier. By means of process transformation rules from [4] any set of syntactically-correct process equations may be transformed to normal form. The concept of machine state space is introduced as a cartesian product of the set of process identifiers with the type of the indexing parameters. We continue to denote elements of this state as $I(v)$. For each event a there can be some pairs of states $(I_i(v), I_i'(v'))$, $i = 1 \ldots k$, so that there is a part of process description of the form

$$I(v) = \ldots \,[]\quad \text{IF } G(v) \text{ THEN } a.i?y!k \to I'(V(y)) \quad []\, \ldots, \tag{1}$$

where i, y, k – dot, input and output parameters of event a, respectively, $G(v)$ holds and $v' = V(y)$. This means that machine makes a transition from the state $I_i(v)$ to the state $I_i'(v')$. The B-machine contains state variables (v) corresponding to the list of indexing parameters and variable p typed over the set of process identifiers. For event a the B-machine contains an operation of the form

$$z \leftarrow a(x, y) \mathrel{\widehat{=}} S_a,$$

where S_a is constructed so that for each occurence of event a in a normalised CSP process the equation of the form (1) in the B operation acquires a SELECT branch of the form:

$$\text{SELECT } G(v) \wedge p = I \wedge x = i \text{ THEN } p, v, z := I', V(y), k \text{ END}$$

All the branches of S_a are composed using the B choice operator $S \,[]\, T$. The language core includes only a limited subset of CSP operators. Complete syntactic definition of the process specification language (the core and its extension) is given in Appendix.

An approach for the mapping of extensions of the core CSP machines into AMN and algorithms of their conversion into B machines have been developed. The algorithms and mechanisms of basic operators conversion are used in accordance with their definition in [4] as soon as a basic operator is detected.

Operators of the core extension due to the complexity of their semantics cannot be transformed to the normal form or reduced to the core operators compositions. Due to that the approach proposed is based on an idea of conversion of a CSP construct into a core CSP construct and a B-machine construct that together preserve the semantics of the original CSP construct. Decomposition of the CSP-text into constructs processable in one such transformation (one step) is specific for each operator of the core extension. The examples of B-machine constructs that can be generated in such step-by-step transformations are new state or flag variables, new SELECT branches in the bodies of B-machine operations: the operations correspond to events and SELECT branches correspond to state transitions. On each step equivalent transformations of CSP terms can be also applied in accordance with the CSP rewriting rules.

During such step-by-step transformation performed by the algorithms the original CSP machine shrinks and the generated B machine is inflated. On each step the semantics of the original CSP machine is preserved by the CSP machine remained after the last transformation, B machine extended during the last transformation and the "cursor" referrring to a position in the remainder of the CSP machine where the next transformation step should start. The transformation terminates when the process description of CSP machine shrinks to an empty sequence of symbols.

The next section shows the mechanisms of transformation of the sequential composition CSP operator into AMN in accordance with the approach mentioned. All other operators of the core extension are provided with similar definitions.

3 Sequential composition operator

STOP is defined as a process executing no actions. The process can terminate *successfully* if it has executed all that it had been committed to do. To distinguish such termination from STOP, it is worthwhile to consider it as a special event *Success*. Process is called *sequential* if it has the *Success* event in alphabet. *Success* can only end the process, that is why *Success* can not serve as an alternative to multiple choice:

$$(x : B \to P(x)) \quad \text{incorrect if} \quad \mathit{Success} \in B$$

SKIP is defined as a process that terminates successfully and does not do anything else: $\mathit{Protocols}(\mathit{SKIP}) = \{<>, < \mathit{Success} >\}$. Here, the brackets $<>$ mean the sequence of events – protocol of the process. While designing a process for certain hard problem solving it may be reasonable to split it into two subproblems, one of which successfully terminates before the beginning of the other. Let P is a sequential process, then sequential composition $(P;\ Q)$ is a process behaving first as P and after its successful termination behaving as Q. If successful termination of P does not occur then $(P;\ Q)$ is not terminated. With a sequential operator it is possible to solve successfully the problem of process parallelizing. Certain stage of the process can reserve huge amount of resources. Parallelizing and futher joining of all parallel subprocesses into one thread may be required. For correct using of sequential operator $(P;\ Q)$ all the branches of P except circles and STOP-ending must terminate with SKIP.

The algorithm of processing of the term $(P;\ Q)$ which is a part of the *ProcessDescription* of some process U is considered further.

PROCESS $S_U = U$ WHERE
ProcessDescription

The algorithm transforms the term according to the CSP rewriting rules [8] to eliminate the sequential composition operator. The algorithm can call itself recursively for internal term, generate new process states and add some SELECT branches to the bodies of operations corresponding to the events. Also the algorithm can call other independent algorithms: for example, the algorithm of assignment operator processing and **ParallelComposition**$(R \parallel S)$ – the algorithm of arbitrary level parallel composition processing.

SequentialOperator$(P;\ Q)$
Start analyzing the first operand.
if $P =$ STOP **then end**
according to the law STOP ; $Q =$ STOP
if $P =$ SKIP **then end**
according to the law SKIP ; $P = P$
if $P = (R;\ S)$ **then**

SequentialOperator$(R;\ (S;\ Q))$ **end**
according to the law $(P;\ Q);\ R = P;\ (Q;\ R)$
if $P = (e : B \rightarrow R(e))$ **then**
SequentialOperator$(R(e);\ Q)$ for all $e \in B$ **end**
according to the law $(e : B \rightarrow R(e));\ Q = (e : B \rightarrow R(e);\ Q)$
if $P =$ (IF G THEN R ELSE S) **then**
SequentialOperator$(R;\ Q)$
SequentialOperator$(S;\ Q)$ **end**
according to the law
(IF G THEN R ELSE S); $Q =$ IF G THEN $(R;\ Q)$ ELSE $(S;\ Q)$
if $P = (R \sqcap S)$ **or** $P = (R \,\Box\, S)$ **then**
SequentialOperator$(R;\ Q)$
SequentialOperator$(S;\ Q)$ **end**
according to the laws
$(R \sqcap S);\ Q = (R;\ Q) \sqcap (S;\ Q)$
$(R \,\Box\, S);\ R = (R;\ Q) \,\Box\, (S;\ Q)$
if $P = (R \parallel S)$ **then**
add the string $I = Q$ to the *ProcessDescription* of process U where I is a new process identifier and a state of process U, then apply the algorithm of processing the parallel composition **ParallelComposition**$(R \parallel S)$ **end**
if $P = (x := e;\ R)$ **then**
apply the algorithm of placing the assignment into the R term, and then apply **SequentialOperatorProcessing**$(R;\ Q)$ **end**
The specific case is
if $Q =$ SKIP **then end**
according to the law $P;$ SKIP $= P$
The last case in analyzing P is
if $P = J$ where J is a process identifier **then**
substitute the current occurence of $(J;\ Q)$ with J, add the string $I = Q$ to the *ProcessDescription* of U. Here I is a new process identifier that expresses a new state of the machine. Continue with the analysis of the process term Q and when a sequential operator occurs inside of it, apply again **SequentialOperator** to the term Q.
if there is no occurence of sequential operator inside of Q **then**
Consider a process P_J given by the J identifier. This identifier is also expresses a state of the machine. While processing the P_J term, store the states from which the machine makes a transition to SKIP. These states are denoted as $S_1 \ldots S_n$. If SKIP is prefixed by event a and (or) is accompanied by a condition G

$$a \rightarrow (\text{ IF } G \text{ THEN SKIP })$$

then such expression is substituted with the

$$a \rightarrow S_j$$

where S_j is a new state of the machine, and the following SELECT branch is added to the *Success* operation of the B machine:

$$\text{SELECT } G \wedge S_M = S_j \text{ THEN } S_M := I \text{ END },$$

where S_M is a variable of the machine state. If a transition into SKIP is not accompanied by a conditional operator then a branch

$$\text{SELECT } S_M = S_k \text{ THEN } S_M := I \text{ END}$$

is added to *Success*. Different branches corresponding to the states S_i are composed using the AMN choice operator [].

end

By its nature, the *Success* event is common for all sequential processes defined in a specification. This is why, while using the sequential operator for description of interleaving processes [4, 8], for example

PROCESS $S_P = |||v : T.P[v]$ WHERE
  $P[v] = a \rightarrow$ SKIP ; S
END

we should parameterize *Success* operation by interleaving index v: $Success(v)$. If there exists yet another interleaving composition process S_Q in a specification indexed by v' then we should parameterize *Success* by v' too. For this case the precondition of *Success* should allow for the indexes to take a special value $NULL$:

$Success(v, v') \mathrel{\widehat{=}}$
  PRE $v \in V \cup \{NULL\} \wedge v' \in V' \cup \{NULL\}$
    SELECT $v \neq NULL \wedge S_P(v) = I$ THEN $S_P(v) = J$ END
    SELECT $v' \neq NULL \wedge S_Q(v') = I'$ THEN $S_Q(v') = J'$ END
    ...
  END

Thus each of the SELECT branches will change a state of only one of the processes taken from the interleaving composition.

4 An example of process specification

To demonstrate capabilities of extended notation, we present a part of lift system specification, a widely used example of modelling concurrent reactive systems

[13]. Full specification uses the sequential composition operator, internal channel communication and real-time aspects of communication. Because of the limited size of the paper, only a part of the specification – *Door* of the lift – will be presented (section 4.5.1 of [13]). The example illustrates the sequential composition, interrupt and $\mu-$operator. *DoorMess* is the set of messages which may be set on the channels from alphabet of *LiftSystem* CSP-machine.

MACHINE *LiftSystem*
SETS $DoorMess = \{ToOpen, ToClose, Opened, Closed, Interrupt\}$
ALPHABET
$Open \quad Close \quad Conf$
$dm \leftarrow Servo$
$Sensor(dm : DoorMessage)$

PROCESS $SDoor = Door$
WHERE
$Door = \mu\, D \bullet Open \rightarrow CycleDoor;\ Close \rightarrow D$
$CycleDoor = Servo!ToOpen \rightarrow OpenDoor;\ Conf \rightarrow$
$(\mu\, CD \bullet (Wait \rightarrow CloseDoor \triangledown (Sensor.Interrupt \rightarrow$
$Servo!ToOpen \rightarrow OpenDoor;\ CD)))$
$OpenDoor = Sensor.Opened \rightarrow$ SKIP
$CloseDoor = Servo!ToClose \rightarrow Sensor.Closed \rightarrow$ SKIP
END
END

The door interfaces on a channel *Servo* with a servomechanism that activates the door to open or close and on a channel *Sensor* to determine when the door is open, closed or blocked from closing. To simplify the *Door* specification, timing aspects were eleminated. Therefore, *Wait* event means the period during which the door must remain open after opening. The safety property of the door is that if the closing of the door is blocked (as indicated by receipt of an *Interrupt* message) then the door must be reopened.

The door cycle is initiated by receipt of an *Open* signal and completed by the *Close* signal. As soon as the door is opened, a *Conf* signal must be sent to indicate fulfillment of a service request. In the *CycleDoor* process the CSP interrupt primitive is used. The normal door cycle follows the order of open-door, wait, then close-door. This normal cycle can be interrupted and restarted by the event which detects a message *Interrupt* from the channel *Sensor*. This message acts as an interrupt of the normal cycle and control is returned to the start of the normal cycle.

Now we consider the mapping of *LiftSystem* CSP-machine into the *LiftSystem* standard B-machine.

MACHINE *LiftSystem*
SETS
$DoorMess = \{ToOpen, ToClose, Opened, Closed, Interrupt\};$

$SDoorState = \{Door, Door1, CycleDoor, CycleDoor1, CD,$
$OpenDoor, OpenDoor1, CloseDoor, CloseDoor1, CloseDoor2, SensorInterrupt\}$
VARIABLES
$SDoor, ValidDoorOpenDoor$
INVARIANT
$SDoor \in SDoorState \wedge$
$ValidDoorOpenDoor \in SDoorState$
INITIALISATION
$SDoor := Door \parallel$
$ValidDoorOpenDoor := CycleDoor$
OPERATIONS
$Success \mathrel{\widehat{=}}$
SELECT $SDoor = OpenDoor1$ THEN
$SDoor := ValidDoorOpenDoor$
WHEN $SDoor = CloseDoor2$ THEN
$SDoor := Door1$
END ;

$Open \mathrel{\widehat{=}}$
SELECT $SDoor = Door$ THEN
$SDoor := CycleDoor$
END ;

$dm \leftarrow Servo \mathrel{\widehat{=}}$
SELECT $SDoor = CycleDoor$ THEN
$SDoor := OpenDoor \parallel dm := ToOpen \parallel$
$ValidDoorOpenDoor := CycleDoor1$
WHEN $SDoor = SensorInterrupt$ THEN
$SDoor := OpenDoor \parallel dm := ToOpen \parallel$
$ValidDoorOpenDoor := CD$
WHEN $SDoor = CloseDoor$ THEN
$SCloseDoor := CloseDoor1 \parallel dm := ToClose$
END ;

$Sensor(dm) \mathrel{\widehat{=}}$
PRE $dm \in DoorMessage$ THEN
SELECT $SDoor = OpenDoor \wedge dm = Opened$ THEN
$SDoor := OpenDoor1$
WHEN $dm = Interrupt$ THEN
$SDoor := SensorInterrupt$
WHEN $SDoor = OpenDoor \wedge dm = Opened$ THEN
$SInterrupt := OpenDoor1$
WHEN $SDoor = CloseDoor1 \wedge dm = Closed$ THEN
$SDoor := CloseDoor2$
END
END ;

$Conf \mathrel{\widehat{=}}$
SELECT $SDoor = OpenDoor1$ THEN
$SDoor := CD$
END

$Close \mathrel{\widehat{=}}$
SELECT $SDoor = Door1$ THEN
$SDoor := Door$
END

$Wait \mathrel{\widehat{=}}$
SELECT $SDoor = CD$ THEN
$SDoor := CloseDoor$
END

END

Some details of mapping of the interrupt operator require additional comments. The interrupt process $I = P \nabla (e \rightarrow Q)$ behaves as P while termination of P or occuring the event e. If event e occures, the control transfers to process Q. So, when *Sensor.Interrupt* occures, system transfers to *SensorInterrupt* state.

Another aspect of parallel composition processing associated with the sequential composition is the following. We reuse the *OpenDoor* process two times and to determine where a control should be transferred after the successful termination of the process *OpenDoor* the variable *ValidDoorOpenDoor* should be created. Before transferring the control to the process *OpenDoor*, a state, where the process should be transferred after the successful termination of *OpenDoor* is assigned to this variable.

5 Specification refinement

As it was mentioned in the Introduction, compositional development of IS is based on specific calculus [12] that provides for identification of fragments of type definitions common for specifications of requirements and specifications of components and for compositions of such fragments refining the specifications of requirements. The notion of provable refinement is crucial for the compositional development. For nonconcurrent specifications the AMN and B-technology provide the required facilities.

The same technology is applicable also to the AMN extended with the process specification facilities introduced in this paper. The tool based on the algorithms developed for the extended process specifications should convert an abstract process description in terms of the extended notation (CSP-machine) into a standard B-machine. The events from the process alphabet are converted into the B-machine operations. Sets, variables, invariants used for a process description are defined by means of typed set theory, as in a standard B-machine. The operations, generated as a result of such conversion, are the generalized substitutions [1]. The process description may contain the operations of B-machines, specified in CONJOINS clause as events. So, the functionality of pre-existing B-machines may be used, and the order of occuring events will be constrained by the process logic.

The consistency of a B-machine obtained after conversion of a process description can be proved with the help of B-Toolkit [2]. The consistency means existence of a model for the machine specification and preserving the invariant by initialisation and operations of the machine.

For the consistent process description the specification refining the process (REFINEMENT) can be written. REFINEMENT provides more concrete definition of the process: e.g., new events making the process more detailed can appear. Events from the initial alphabet can be implemented by means of operations of existing B-machines.

Relation between the variables of the B-machine, obtained as a result of conversion from the process description, and the variables of the refining B-machine should be defined in the invariant of the REFINEMENT machine. Generally, while refining CSP-machines (process descriptions) first of all it is necessary to convert CSP-machine and its REFINEMENT into B-machines. After that, the state space of the process and its refinement and the set of transitions between such states will be defined. Then the invariant for the refinement should be written defining the relation between state spaces of the machines [4]. This invariant should be added to the CSP-refinement and the CSP-refinement should be converted into the B-machine once more. After that the correctness of the refinement can be proved with the conventional B-Toolkit facilities.

The IMPLEMENTATION specifications of AMN (the final refinement) can also be used. The IMPLEMENTATION may not contain its own variables, parallel substitutions, nondeterminism, operation preconditions. All the data stuctures and functionality are imported from the other B-machines. Thus, the system can be constructed from components (imported B-machines) and the fact of refining the initial specification by the IMPLEMENTATION would be justified if all theorems, generated by B-Toolkit in the process of refining were proved.

6 Related work

The aim of this work is to present a notation capable of addressing both data-algorithmic issues and process control issues. The technology for getting a proof of refinement correctness in such notation is critical for the compositional IS develpment.

The core of the process specification notation proposed is based on the subset of CSP used in csp2B [4, 5]. The idea of CSP-machines mapping to AMN is similar to those of [4, 5].

The extension of the core proposed has been influenced by TCOZ [13] approach of integrating the Object Z notation [17, 6] and Timed CSP [15, 16]. The power of this notation in modelling concurrent reactive systems allows to treat it as a reference model: any notation may be considered to be well-expressive if it supports the most of the TCOZ facilities. To demonstrate the expressive power of extended AMN, the lift system example from [13] was chosen. For this example the extended AMN specification and the respective B-machine has been constructed.

7 Conclusions

The work presents an extension of AMN with the process specification facilities. AMN constructs into which the process specifications are mapped and algorithms of process specifications conversion into standard B-machines are introduced. Additionally to the core (csp2B), the extended notation supports the sequential processes, parallel composition on arbitrary level, hiding, nondeterministic and general choice, multiple choice, interrupt operator, assignment operator, timing primitives – delay, timeout and time prefixing. While specifying processes, it is possible to use the expressions, types and predicates of standard AMN and to use the operations of another B-machines as events. Thus, the integration of extended and standard notation is reached.

The interpretation of process description operators allows to constuct the refinements of process specifications and to prove the correctness of refinement using B-technology. This provides a basis required for compositional IS development.

The extended AMN notation allows to model the reactive concurrent systems operating with complex data stuctures (workflows constitute important example of such systems). Capabilities of the extended notation are illustrated by the lift system example showing how to define clearly the process logic in the process part of specification.

References

[1] J. -R. Abrial. The B-Book. Cambridge University Press, 1996.

[2] B-Toolkit User's Manual - Release 3.4. Copyright B-Core (UK) Ltd., 1997.

[3] D. O. Briukhov, L. A. Kalinichenko. Component-Based Information Systems Development Tool Supporting the SYNTHESIS Design Method. Proc. of the East European Symposium on "Advances in Databases and Information Systems", Poland, Springer, LNCS No.1475, 1998

[4] M. J. Butler. csp2B: A Practical Approach To Combining CSP and B. Declarative Systems and Software Engineering Group, Technical Report DSSE-TR-99-2, February 1999.

[5] M. J. Butler. csp2B User Manual - Version 1.1. Available from http://www.ecs.soton.ac.uk, 1999.

[6] R. Duke, P. King, G. Rose and G.Smith. The Object-Z specification language: Version 1. Technical Report 91-1, Software Verification Research Centre, Australia, 1991

[7] A. Elmagarmid and W. Du. Workflow Management: State of Arts vs. State of Market. Advances in Workflow Management Systems and Interoperability, Istanbul, August 1997.

[8] C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.

[9] Kalinichenko L.A. SYNTHESIS: the language for desription, design and programming of the heterogeneous interoperable information resource environment. Institute for Problems of Informatics, Russian Academy of Sciences, Moscow, 1995, 103 p.

[10] Kalinichenko L.A. Workflow Reuse and Semantic Interoperation Issues. In *Advances in workflow management systems and interoperability.* A.Dogac, L.Kalinichenko, M.T. Ozsu, A.Sheth (Eds.). NATO Advanced Study Institute, Istanbul, August 1997

[11] Kalinichenko L.A. Composition of type specifications exhibiting the interactive behaviour. In *Proceedings of EDBT'98 Workshop on Workflow Management Systems,* March 1998, Valencia

[12] L. A. Kalinichenko. *Compositional Specification Calculus for Information Systems Development.* In Proc. of the East-West Symposium on Advances in Databases and Information Systems (ADBIS'99), Maribor, Slovenia, Lect. Notes Comput. Sci., 1999

[13] B. Mahony and J. S. Dong. Timed Communicating Object Z. IEEE Transactions on Software Engineering, Vol. 26 (2), Feb 2000.

[14] R. Milner. Communication and Concurrency. Prentice-Hall, 1989.

[15] S. Schneider. An Operational Semantics for Timed CSP. Information and Computation, 116(2), 1995.

[16] S. Schneider and J. Davies. A Brief History of Timed CSP. Theoretical Computer Science, 138, 1995.

[17] J. Spivey. The Z Notation: A Reference Manual (2nd Ed.). International Series in Computer Science. Prentice Hall, 1992.

[18] *Workflow Management Coalition,* http://www.aiai.ed.ac.uk:80/WfMC

[19] *Workflow Process Definition,* RFP, OMG, http://cgi.omg.org/cgi-bin/doc?bom/2000-03-03

Appendix: Process description syntax rules

In this section, $t \mid s$ indicates t or s, $< t >$ indicates zero or one occurences of t, $<< t >>$ indicates zero or more occurences of t. As a *preamble* csp-machine can contain the next standard B clauses: INCLUDES, USES, SEES, IMPORTS, EXTENDS, PROMOTES, CONSTRAINTS, SETS, DEFINITIONS, CONSTANTS, PROPERTIES, VARIABLES, INVARIANT. Section CHANNELS consists the names of events which are the channels of internal communication of the specification. Section ALPHABET_ SUBSET consists B set definition of alphabet subsets: such subsets may be used in hiding and multiple choice operators. To distinguish the additional operators the **bold comments** are used. For the notation symbols $\sqcap$ $\Box$ $\triangledown$ $\triangleright$ μ $\bullet$ the corresponding ASCII combinations will be used, respectively: $|-|$ $|\_|$ $\backslash - /$ $|>$ MU ..

machine ::=
MACHINE *ident* < (*ident_list*) >
preamble
< CONJOINS *ident* < (*exp_list*) >>
< CHANNELS *alpha_list* >
ALPHABET *alpha_list*
< ALPHABET_ SUBSET << *ident* = {*ident_list*} >> >
processes
END

processes ::= *process* << *process* >>
process ::=
PROCESS *ident* =< |||*ident_list* : *explist*. > *ident* < [*ident_list*] >< (*exp_list*) >
< CONSTRAINS *constrain_list* > WHERE
<< *ident* < [*ident_list*] < (*typed_ident_list*) >= *proc*0 >>
END

recurse ::= *ident* < [*ident_list*] >< *exp_list* >
recurse_list ::= *recurse* <<, *recurse* >>
event ::= *ident* << .*exp* |?*ident* >><<!*exp* >>
constrain_list ::= *constrain* << *constrain* >>
constrain ::=< *ident* ←| (*ident_list*) ←> *ident* < (*ident_list*) >
set_expr ::= *ident* | B set definition
typed_ident_list ::= *ident* : *exp* <<, *ident* : *exp* >>
ident_list ::= *ident* <<, *ident* >>
ident ::= alphnumeric identifier starting with alpha character
exp_list ::= *exp* <<, *exp* >>
exp ::= any valid B expression or predicate

*proc*0 ::= *proc*1 | *proc*4
*proc*1 ::= *proc*2 << [] *proc*2 >>
*proc*2 ::=
STOP
| IF *exp* THEN *proc*2 < ELSE *proc*2 >
| (*proc*1)

	$\mid \quad event \rightarrow proc3$
multiple choice	$\mid \quad (ident : set\_expr \rightarrow proc2)$
time prefixing	$\mid \quad event \bullet ident \rightarrow proc3$
	$proc3 ::= proc2 \quad \mid \quad recurse \mid \quad proc5$
	$proc4 ::=$
nondeterministic choice	$proc5 << \sqcap proc5 >>$
general choice	$\mid \quad proc5 << \Box proc2 >>$
hiding	$\mid \quad proc5 \setminus set\_expr$
assignment	$\mid \quad ident := exp;\ proc5$
sequential composition	$\mid \quad proc5 <<;\ proc3 >>$
μ −operator	$\mid \quad \mu\, ident \bullet proc5$
delay	$\mid \quad \text{WAIT } ident;\ proc5$
interrupt	$\mid \quad proc5 \nabla (event \rightarrow proc3)$
timeout	$\mid \quad (event \rightarrow proc3) \triangleright \{ident\} proc5$
	$\mid \quad parallel$
	$proc5 ::= (proc4) \quad \mid \quad proc2 \quad \mid \quad \text{SKIP}$
parallel composition	$parallel ::= \parallel (recurse\_list) \text{ WHERE } processes$