29

CSP-OZ: A Combination of Object-Z and CSP

Clemens Fischer
University of Oldenburg, Department of Computer Science
P.O. Box 2503, 26111 Oldenburg, Germany.
E-mail: Fischer@Informatik.Uni-Oldenburg.de

Abstract

In this paper we define a combination of Object-Z and CSP called CSP-OZ. The basic idea is to define a CSP-semantics for every Object-Z class. Special care is taken to capture the characteristics of input and output parameters properly and to preserve the expected refinement rules.

CSP-OZ is well suited for the specification and development of communicating distributed systems. It provides powerful techniques to model data- and control-aspects in a common framework. The language is easy to use for Z and Object-Z users.

Keywords

Z, Object-Z, CSP, concurrent systems, combining FDTs, refinement.

1 INTRODUCTION

For the definition of the semantics of parallel, object-oriented languages, two things must be considered: On the one hand structuring features like inheritance are very important (*internal view*) and on the other hand the dynamic behaviour of objects executed in parallel has to be defined (*external view*). It is convenient to separate these aspects because an external observer of the dynamic behaviour of an object should not be able to look into the internal structure. This paper solely deals with the external view of an object; i.e. the dynamic semantics of objects.

Object-Z [6] is an object-oriented extension of Z [22] for the predicative specification of objects. An Object-Z class consists of the specification of a

*This research is supported by the German Ministry for Education and Research (BMBF) as part of the project UniForM under grant No. FKZ 01 IS 521 B2.

state space and operations on this state space. The process algebra CSP [12] has been developed to describe the dynamic behaviour of a system and covers aspects like parallel composition, hiding and divergence.

The basic idea we follow in this paper was suggested by Smith [21]: We define the dynamic semantics of an Object-Z class using the semantic model of CSP. Thereby all CSP-operators like parallel-composition and hiding can be applied to objects.

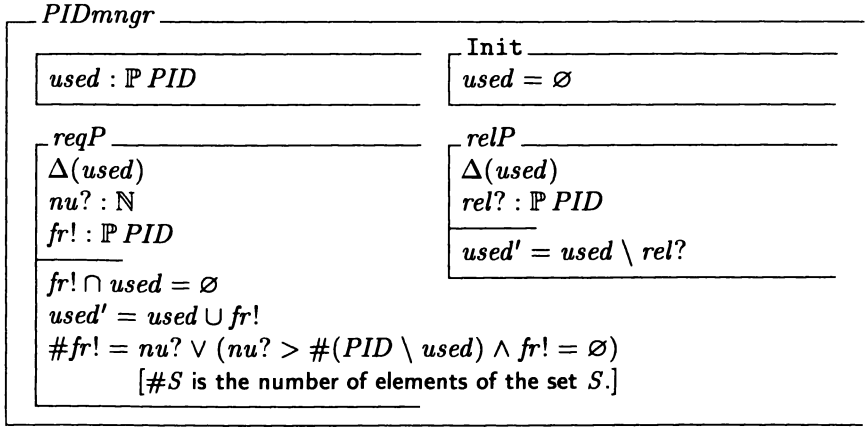
This work is in the same line as [7], where CSP and Z are combined in a similar fashion. A major difference between [21] and [7] is the handling of input and output parameters of an operation. In [21] no semantic difference is made between input and output parameters, whereas in [7] the difference between input and output is captured semantically, but the mixture of input and output parameters is not considered yet.

The contribution of this paper is to define a CSP failure-divergence semantics of Object-Z integrating the views of [21] and [7] and extending them to mixtures of input and output parameters. This combination of Z, Object-Z and CSP is called CSP-OZ.

The rest of this paper is organised as follows. The ideas of the semantics are presented in the next section with an example. The version of Object-Z used here and the semantic model of CSP are introduced in the subsequent sections. In section 5 the failure-divergence semantics of CSP-OZ is defined. Finally, a conclusion is drawn and connections to related work are discussed.

2 ILLUSTRATING EXAMPLE

To illustrate the approach, we present a system for the management of free and used process identifiers (PIDs): $PID == 1 \,..\, maxP$ where $maxP : \mathbb{N}_1$ is a global variable. The Object-Z specification of the system is the class *PIDmngr*.



The unnamed schema in the left corner defines the state space of the class. We use the keyword `State` to refer to this schema. The variable *used* stores the

set of PIDs that are used by some process in the environment of *PIDmngr*. The schema `Init` specifies the initial values of *used*. The class has two operations: *reqP* and *relP*. The variables declared in the delta list ($\Delta(used)$) can be changed by the operation. The parameters of *reqP* are *nu*? and *fr*!. The decorations ? and ! are used for inputs and outputs, respectively. The schema *reqP* describes how the state of *PIDmngr* changes when *reqP* is applied. The undecorated variable *used* corresponds to the state before the operation and the dashed variable *used'* is the final value. Hence *reqP* computes a set of unused PIDs containing *nu*? elements. If there are not enough free PIDs available, the empty set is returned.

The basic idea of CSP-OZ is to define the semantics of a class using the semantic model of CSP. Thereby all CSP-operators – especially hiding and parallel composition – can be used to combine objects, and CSP and Object-Z syntax can be mixed.

The CSP-semantics is based on the alphabet of a process; i.e. the set of events the environment can observe. The observable events of a class are the operations with their parameters [20, 21, 7]. For example, $(reqP, \{nu \mapsto 3, fr \mapsto \{2,4,5\}\})$ and $(relP, \{rel \mapsto \{2,5,6\}\})$ are possible events of *PIDmngr* (provided $maxP \geq 6$). The set $\{rel \mapsto \{2,5,6\}\}$ denotes the function where *rel* is mapped to the set $\{2,5,6\}$.

To make these events explicit, we extend Object-Z with the declarations of channels. The schema names of the operations corresponding to a channel are prefixed with the keyword `com` (more keywords will be introduced later). The CSP-OZ specification of *PIDmngr* is the following class:

S-PID$_1$

```
channel reqP : [ nu : ℕ; fr : ℙ PID ]
channel relP : [ rel : ℙ PID ]
PIDmngr[com_reqP/reqP, com_relP/relP]
```

The specification *S-PID*$_1$ inherits all schemas of *PIDmngr*, but *reqP* and *relP* are renamed to `com_reqP` and `com_relP`, respectively. The declaration of *PIDmngr* in *S-PID*$_1$ can textually be replaced by the definition of *PIDmngr*.

The semantics of a CSP-process is given in terms of the *failures* and the *divergences*. A failure is a tuple (tr, X) consisting of a trace tr of events the process may perform and a set of events X the process can refuse to engage in after tr. The divergences are a set of traces after which future behaviour of the process is unpredictable.

A class C can engage in the trace $\langle e_1, \ldots, e_n \rangle$ of events $e_i = (op_i, p_i)$ if the successive application of the operation op_i with parameter p_i transfers some initial state of C into some reachable state. The trace

$$tr_1 = \langle\ (reqP, \{nu \mapsto 3, fr \mapsto \{2,4,5\}\}),\quad (reqP, \{nu \mapsto 1, fr \mapsto \{8\}\}),\quad (relP, \{rel \mapsto \{2,4,8\}\})\rangle \tag{1}$$

transfers the initial state of *S-PID*$_1$ into the final state $used = \{5\}$.

The crucial question we deal with in this paper, is the definition of the refusals of a class. To understand the problem, we have to look at the CSP-refinement relation. A process P_1 *refines* (or correctly implements) a process P_2 if $\mathcal{F}(P_1) \subseteq \mathcal{F}(P_2)$ (P_1 is more deterministic) and $\mathcal{D}(P_1) \subseteq \mathcal{D}(P_2)$ (P_1 is less divergent; i.e. more defined), where $\mathcal{F}$ and $\mathcal{D}$ denote the set of failures and divergences of a process, respectively. CSP-refinement is compositional, i.e. whenever P_1 refines P_2 then $C(P_1)$ refines $C(P_2)$ for any system context $C(\cdot)$. We expect that the following class is an implementation of *S-PID*$_1$.

I-PID

$\texttt{channel}\ reqP : [\, nu : \mathbb{N};\ fr : \mathbb{P}\, PID\,]$

$\texttt{channel}\ relP : [\, rel : \mathbb{P}\, PID\,] \qquad \texttt{Init} \mathrel{\widehat{=}} [\forall p : PID \bullet u(p) = 0]$

$u : PID \to \{0, 1\}$ [u is used to store the status of every PID.]

$next : 1\,..\,maxP + 1$ [$next$ is a pointer to the smallest free PID.]

$$next = min(\{p : PID \mid u(p) = 0\} \cup \{maxP + 1\})$$

com_*reqP*

$\Delta(u, next)$

$nu? : \mathbb{N}$

$fr! : \mathbb{P}\, PID$ [$fr!$ is the set of free PIDs starting with next.]

$$nu? \leq maxP + 1 - next \Rightarrow (\#fr! = nu? \wedge min\, fr! = next \wedge \forall p : fr! \bullet u(p) = 0 \wedge u'(p) = 1 \wedge max\, fr! = next' - 1 \wedge \forall p : PID \setminus fr! \bullet u'(p) = u(p))$$

$$nu? > maxP + 1 - next \Rightarrow (\#fr! = \varnothing \wedge u' = u)$$

com_*relP*

$\Delta(u, next)$

$rel? : \mathbb{P}\, PID$

$$\forall p : rel? \bullet u'(p) = 0 \wedge \forall p : PID \setminus rel? \bullet u'(p) = u(p)$$

The behaviour of *I-PID* differs from *S-PID*$_1$ in one aspect: A request for new PIDs is answered with the set of PIDs starting with *next*, not with an arbitrary set of free PIDs. E.g. if no PIDs are used, any request for three PIDs is answered with the set $\{1, 2, 3\}$, whereas *S-PID*$_1$ could send back any set of three numbers.

This is an acceptable behaviour of an implementation because *fr* is declared as an output parameter. The specification *S-PID*$_1$ makes a nondeterministic choice between different values of *fr*. Any implementation can reduce this nondeterminism. Following this idea, *S-PID*$_1$ can refuse the following sets of

events initially (i. e. before engaging in any event):

$$R_1 = \{X : \mathbb{P}\, Event \mid (\forall n : 0\,..\,maxP \bullet \exists f : \mathbb{P}\, PID \bullet \#f = n \wedge (reqP, \{nu \mapsto n, fr \mapsto f\}) \notin X) \wedge \quad (2)$$
$$(\forall n > maxP \bullet (reqP, \{nu \mapsto n, fr \mapsto \varnothing\}) \notin X) \wedge \quad (3)$$
$$(\forall e : X \bullet first(e) \neq relP)\ \} \quad (4)$$

To understand this definition, think of the events *S-PID*$_1$ must engage in. Consequently (2) - (4) says something about events that cannot be refused. In (2) the property is captured that *S-PID*$_1$ must deliver some set of PIDs if n is not larger than *maxP*. The existential quantification $\exists f : \mathbb{P}\, PID$ captures the nondeterministic choice of possible output parameters. Line (3) deals with n exceeding *maxP*. Then *S-PID*$_1$ does not have a choice for *fr*; the empty set is always returned. Finally, *S-PID*$_1$ can never refuse a communication on *relP* for every choice of the parameter *rel* (4). The function *first* computes the first component of the tuple e (i. e. the channel of e).

Recall that we just considered initial refusals, but the calculation of the refusals after some trace *tr* is similar. The initial refusals of *I-PID* are

$$R_I = \{X : \mathbb{P}\, Event \mid (\forall n : 0\,..\,maxP \bullet (reqP, \{nu \mapsto n, fr \mapsto \{1, \ldots, n\}\}) \notin X) \wedge \quad (5)$$
$$(\forall n > maxP \bullet (reqP, \{nu \mapsto n, fr \mapsto \varnothing\}) \notin X) \wedge \quad (6)$$
$$(\forall e : X \bullet first(e) \neq relP)\ \}. \quad (7)$$

Line (6) and (7) are the same as (3) and (4), but the existential quantification in (2) is removed in (5). This models the fact that the nondeterminism is removed in *I-PID*. It initially always returns $\{1, \ldots, n\}$ when n PIDs are requested. The initial refusals of *I-PID* are a subset of the initial refusals of *S-PID*$_1$ as expected. This is also true for the refusals after engaging in any possible trace. Thus *I-PID* is an implementation of *S-PID*$_1$ in the failure-divergence semantics.

However, this is not the case if we follow the standard Object-Z semantics where no difference is made between input and output parameters [6]: Any parameter can be controlled by the environment *and* the system. Consequently, *I-PID* is not an implementation of *S-PID*$_1$ according to [21], because the choice between the different values for *fr*! would not be made nondeterministically.

Nevertheless, the blocking view of an operation can be useful for other examples and we integrate it into CSP-OZ. In the following version of *PIDmngr* the parameter *nu* of *reqP* is used without decoration. We call this a *simple* parameter. Its value is under the control of the object *and* the environment.

S-PID$_2$

S-PID$_1$[**redef** com_*reqP*]

com_reqP

$\Delta(used)$
$nu : \mathbb{N}$
$fr! : \mathbb{P}\, PID$

$\#fr! = nu \land fr! \cap used = \varnothing \land used' = used \cup fr!$

The keyword **redef** indicates that $S\text{-}PID_2$ inherits $S\text{-}PID_1$ except for the definition of com_reqP. Instead of sending an empty set of free process identifiers, any unrealizable request for PIDs is blocked in $S\text{-}PID_2$. The initial refusals are the following sets of events.

$$R_2 = \{X : \mathbb{P}\, Event \mid (\forall n : 0 \,..\, maxP \bullet \exists f : \mathbb{P}\, PID \bullet \#f = n \land (reqP, \{nu \mapsto n, fr \mapsto f\}) \notin X) \land (\forall e : X \bullet first(e) \neq relP)\ \}$$

The last question to address in this section is the guard of an operation, i. e. the condition when the operation is enabled. In Z operations are always enabled, but in Object-Z and a lot of interpretations of Z other views are adopted. To integrate these different views and to avoid confusion about the guard, we use an extra schema to determine the guard of an operation. This is neither in the tradition of Z nor of Object-Z, but it is in line with many different languages like action systems, B, VDM and MIX. The following specification can refuse to release PIDs that are not used.

$S\text{-}PID_3$

$S\text{-}PID_2$[**redef** com_relP]

enable_relP

$rel : \mathbb{P}\, PID$

$rel \subseteq used$

effect_relP

$\Delta(used)$
$rel : \mathbb{P}\, PID$

$rel \neq \varnothing \land used' = used \setminus rel$

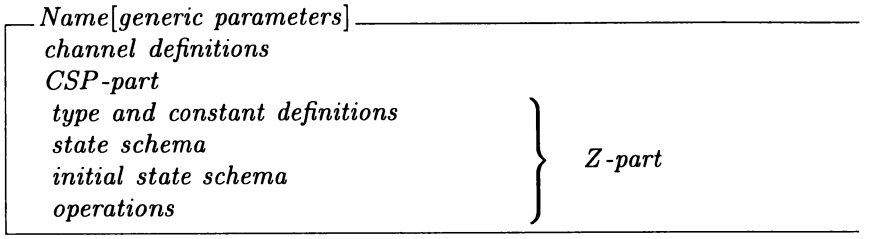
We use the prefixed keywords **enable** for the guard and **effect** for the schema describing the state change of an operation. Communications outside **enable** are blocked. A communication that is not blocked, but that is applied outside the precondition of **effect** leads to divergence. Thus $S\text{-}PID_3$ refuses any communication on *relP* initially. If the empty set is released the system will diverge.

Note that an operation **effect_***c* without simple parameters and without an enable-schema (abbreviates the guard *true*) models the Z-view of an operation. Hence CSP-OZ integrates the non-blocking view traditionally adopted in Z and the blocking view of Object-Z. It even allows arbitrary mixtures of these aspects. Only in this special case the enable schema is necessary.

3 SYNTAX OF CSP-OZ

Object-Z [6] extends Z with the notion of a class, specifying a state space and operations on this state space. In this paper we consider only a restricted version of Object-Z. The parallel and sequential composition operators are no longer used inside Object-Z (the corresponding CSP operators can be used instead); and we do not allow history invariants as they cannot be modelled properly in our semantic model.

CSP-OZ extends Object-Z with the notion of communication channels and CSP-syntax. A CSP-OZ class has the following basic structure.



The *channel definitions* declare the *interface* of the class. Every declaration has the form **channel** $c : [p_1 : ty_1; \ldots; p_n : ty_n]$ where c is the channel name, $p_1, \ldots, p_n$ is the possibly empty list of undecorated parameter names and $ty_1, \ldots, ty_n$ are the type declarations of the parameters.

The CSP-part is a set of equations of the form *CSP-name* = *CSP-process* where *CSP-process* is defined using the syntax of the CSP model checker FDR [9] extended with Z-types. The channels of every process in the CSP-part must be a subset of the channels declared in the interface. If the CSP-part is not empty a process with the keyword **main** must be defined. This process is used to determine the semantics of the CSP-part. Other CSP-processes can be used to enhance readability of the CSP-part.

The type and constant definitions and the state and initial state schemas of the *Z-part* are the same as in Object-Z [20]. The schema, type and variable names used in the Z-part must be disjoint from the names used in the CSP-part.

For every channel c declared in the interface there must be either the schema **effect_**c and optional the schema **enable_**c or the schema named **com_**c, which is an abbreviation for

$$\mathbf{effect\_}c \mathrel{\widehat=} \mathbf{com\_}c$$
$$\mathbf{enable\_}c \mathrel{\widehat=} \exists \mathtt{State}'; p_{s+1}? : ty_{s+1}; \ldots; p_n! : ty_n \bullet \mathbf{com\_}c.$$

All parameters of c must be declared in **effect_**c together with the possible decorations ? (input) and ! (output). An undecorated parameter stands for a simple parameter. The schema **enable_**c specifies the states and simple parameters in which c cannot refuse to communicate on c. Hence we assume

that `enable_c` and `effect_c` can be normalised to

enable_c

$p_1 : ty_1; \ldots; p_s : ty_s$

P_{ena}

effect_c

$\Delta(v_1, \ldots, v_m)$

$p_1 : ty_1; \ldots; p_s : ty_s$

$p_{s+1}? : ty_{s+1}; \ldots; p_{s+i}? : ty_{s+i}$

$p_{s+i+1}! : ty_{s+i+1}; \ldots; p_n! : ty_n$

P_{eff}

Any of the parameter lists can be empty. If `enable_c` is omitted P_{ena} is set to true. Other schemas – not prefixed with a keyword – may freely be used to structure the specification, but there is always a class without further schemas with the same failure-divergence semantics.

As the Z-part of a CSP-OZ class corresponds directly to an Object-Z class, features like inheritance or instantiation of Object-Z classes can be used to structure a CSP-OZ class. But in this paper we assume that all classes can be normalised to the form given above.

CSP-OZ classes can be combined using the following CSP-operators:

- $\mathcal{C}_1 [| A |] \mathcal{C}_2$ is the parallel composition of $\mathcal{C}_1$ and $\mathcal{C}_2$ synchronising on the set of events A.
- $\mathcal{C} \setminus A$ is the class where all events in the set A are hidden; no $e \in A$ can be observed by the environment.
- $\mathcal{C}_1 \,|||\, \mathcal{C}_2$, $\mathcal{C}_1 \sqcap \mathcal{C}_2$ and $\mathcal{C}_1 \,\Box\, \mathcal{C}_2$ are the interleaving, internal choice and external choice of $\mathcal{C}_1$ and $\mathcal{C}_2$.

The notation $\{|\ relP\ |\}$ denotes the set of events of the channel *relP*. As an example, we specify the registration desk of a hotel. A connection diagram [12] can be found in Figure 1. Every box stands for a CSP-OZ class. The lines between boxes denote hidden channels and the four dangling lines are the observable interface of the specification. An event on the channel *request* indicates that a guest wants to register. Then a new PID is requested and a card with the PID is printed. The printer delivers the card (*card*) or fails to produce a card (*nocard*).

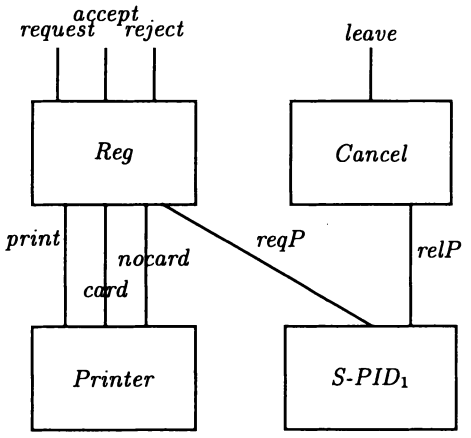


Figure 1 The connection diagram for *RegDesk*

A successful registration is indicated by *accept* and any error in this pro-

cedure results in a communication on *reject*. Leaving the hotel is indicated by a communication on *leave*. Then the PID is released. We omit a detailed specification of *Reg* and *Cancel*. The printer is modelled by the following specification.

Printer

$$\texttt{channel } print : [\, p : PID \,]$$
$$\texttt{channel } card, nocard : [\,]$$
$$\texttt{main} = print?x \rightarrow (card \rightarrow \texttt{main} \;\Box\; nocard \rightarrow \texttt{main})$$

The precise semantics of the connection diagram in fig. 1 is given by the following CSP-expression.

$$RegDesk = \Big(((Reg \,[|\{|\, print, card, nocard \,|\}|]\, Printer) \; [|\{|\, reqP \,|\}|]\; S\text{-}PID_1) \; [|\{|\, relP \,|\}|]\; Cancel \Big) \setminus \{|\, relP, reqP, print, card, nocard \,|\}$$

4 THE SEMANTIC MODEL OF CSP

The standard semantic model of CSP is the failure-divergence model. The semantics given in [12] is restricted to finite alphabets. Straight forward extensions of this model to infinite alphabets are restricted to bounded nondeterminism. To allow unbounded nondeterminism, two models were developed: A new order for the definition of fix points is introduced in [17] and in [18] the failure-divergence model is extended with the set of infinite traces a process may engage in. Generally speaking, both models can be used for the semantics of CSP-OZ. The model in [18] avoids the anomaly that hiding of an event e leads to divergence if it can occur finitely many times without a finite bound for the number of possible occurrences of e. But we prefer here the model of [17] as it is similar to the standard model of CSP [12] and is the basis of a CSP-encoding in the theorem prover Isabelle [24] that is used in the project UniForM [15] to develop tool support for CSP-OZ. Nevertheless, the definitions presented here are also applicable to the infinite trace model and the differences completely vanish if only bounded nondeterminism is considered.

Definition 1 *Let $\mathcal{A}$ be a possibly infinite alphabet of events. Then the semantics of a CSP-OZ specification is a tuple $(\mathcal{F}, \mathcal{D})$ where*

1. *the failure set $\mathcal{F} : seq\mathcal{A} \leftrightarrow \mathbb{P}\mathcal{A}$ is a relation between traces on $\mathcal{A}$ and subsets of $\mathcal{A}$. A tuple (tr, X) is element of $\mathcal{F}$ iff tr is a possible trace and all communications from X can be refused after engaging in tr.*

2. *the divergence set* $\mathcal{D} : \mathbb{P}\, seq\mathcal{A}$ *with* $\mathcal{D} \subseteq dom\mathcal{F}$ *is the set of traces after which the system may diverge,*

The set of $(\mathcal{F}, \mathcal{D})$ *fulfilling the healthiness conditions of [17]* is called* $\mathcal{N}'$. □

5 A FAILURE-DIVERGENCE SEMANTICS OF CSP-OZ

The semantics for CSP is defined in [17]. Thus our main task is the definition of the failure-divergence semantics of CSP-OZ classes with an empty CSP-part. The combination of the CSP-part and the Z-part is defined at the end of this section using the parallel operator.

Let Id_c denote the set of all channel identifiers. Let Id_v denote the set of undecorated variable identifiers and let Id_p denote the set of undecorated parameter names. Finally, let *Value* denote the set of possible values that can be assigned to variables and parameters. The exact definition of Id_c, Id_v, Id_p and *Value* is not formalised here. A state is a finite partial function assigning values to variable-identifiers.

$$State == Id_v \pfun Value$$

An event is a tuple consisting of a channel-identifier and a parameter, which is a finite partial function assigning values to parameter-names.

$$Parameter == Id_p \pfun Value$$
$$Event == Id_c \times Parameter$$

A CSP-OZ class with an empty CSP-part induces a set of channels, states, and initial states; a transition relation; a set of input, output, and simple parameters for every channel and a set of enabled parameters for every state and channel. This is modelled by the schema type *ClassStruct* with the following signature.

ClassStruct

$$states, initial : \mathbb{P}\, State$$
$$channels : \mathbb{P}\, Id_c$$
$$trans : Event \pfun (State \pfun \mathbb{P}\, State)$$
$$simple_p : Id_c \pfun \mathbb{P}\, Id_p$$
$$in_p, out_p : Id_c \pfun \mathbb{P}\, Parameter$$
$$enable : Id_c \times State \pfun \mathbb{P}\, Parameter$$

We only give an informal description of the translation of CSP-OZ syntax to the induced *ClassStruct*. The construction of *states*, *initial* and *channels* for

*The healthiness conditions of [17] rule out miraculous specifications; require a prefix closed set of traces, subset closed refusal sets, and any set of communications that are impossible in the next step are refused. Furthermore, the divergence set must be a suffix closed and the chaotic closure is included in $\mathcal{F}$ in case of divergence.

a given class $\mathcal{C}$ is obvious. We only consider classes where $initial \neq \varnothing$. The set $simple_p(c)$ consists of all simple parameter names declared for the channel c. The sets $in_p(c)$ and $out_p(c)$ contain all functions assigning values to the input and output parameters. The possible values of the parameters are only affected by the declaration of the channel, not by the corresponding schemas. If a channel c does not have input parameters we assume $in_p(c) = \{\varnothing\}$ (output parameters analogously). The set $trans(e)(st)$ consists of all states reachable by applying the event e to the state st. E. g. for $S\text{-}PID_1$ we have

$$trans(reqP, \{nu \mapsto 1, fr \mapsto \{1\}\})(\{used \mapsto \{4\}\}) = \{\{used \mapsto \{1,4\}\}\}$$
$$trans(reqP, \{nu \mapsto 3, fr \mapsto \{1\}\})(\{used \mapsto \{4\}\}) = \{\}$$

The function *enable* computes for a given channel and state the enabled simple parameters. E. g. for $S\text{-}PID_3$ we have

$$enable(relP, \{used \mapsto \{1,3,4\}\}) = \{\{rel \mapsto X\} \mid X \subseteq \{1,3,4\}\}$$

and for $S\text{-}PID_1$ we have $enable(relP, st) = \{\varnothing\}$ for all states st because $relP$ does not have a simple parameter. If $S\text{-}PID_1$ could refuse a communication on $relP$ for some st we would have $enable(relP, st) = \{\}$. This trick of notation simplifies the following formal definitions significantly because we do not have to consider separate cases for channels without all three kinds of parameters.

We define the failure-divergence semantics of CSP-OZ based on *ClassStruct*. The function *reachable* computes the set of reachable states for a trace.

$$reachable : ClassStruct \to (\text{seq } Event \to \mathbb{P}\, State)$$

$$\begin{aligned}
&\forall \mathcal{C} : ClassStruct \bullet \\
&\quad reachable(\mathcal{C})(\langle\rangle) = \mathcal{C}.initial \land \\
&\quad \forall tr : \text{seq } Event \bullet reachable(\mathcal{C})(tr \frown \langle e \rangle) = \\
&\qquad \{st : State \mid (\exists\, st_1 : reachable(\mathcal{C})(tr) \bullet st \in \mathcal{C}.trans(e)(st_1))\}
\end{aligned}$$

Using tr_1 (1) we have $reachable(S\text{-}PID_1)(tr_1) = \{\{used \mapsto \{5\}\}\}$ if we identify $S\text{-}PID_1$ with the induced *ClassStruct*.

The function $\mathcal{D}$ computes the set of traces after which a class may diverge. A trace tr is in $\mathcal{D}(\mathcal{C})$ if it can be divided into $s \frown \langle c, v\rangle \frown t$ where $\langle c, v\rangle$ is the event causing the divergence which means $\langle c, v\rangle$ is enabled after s but no transition is defined for $\langle c, v\rangle$. Because of the healthiness conditions any trace t must be possible after $s \frown \langle c, v\rangle$.

$$\mathcal{D} : ClassStruct \to \mathbb{P}\,\text{seq } Event$$

$$\begin{aligned}
&\forall \mathcal{C} : ClassStruct \bullet \mathcal{D}(\mathcal{C}) = \\
&\quad \{s, t : \text{seq } Event;\ c : Id_c;\ v : Parameter \mid \\
&\qquad \exists\, st : reachable(\mathcal{C})(s) \bullet ((\mathcal{C}.simple_p(c) \lhd v) \in \mathcal{C}.enable(c, st) \land \\
&\qquad\qquad [v \text{ (restricted to the simple parameters) is enabled in } st.] \\
&\qquad\quad \forall o : \mathcal{C}.out_p(c) \bullet \mathcal{C}.trans(c, v \oplus o)(st) = \varnothing) \\
&\qquad\qquad [\text{No transition is defined for any choice of output parameters.}] \\
&\qquad \bullet (s \frown \langle c, v\rangle \frown t)\}
\end{aligned}$$

Recall that $\{D \mid P \bullet E\}$ is the set of all evaluations of the expression E in the context of all variables declared in D and restricted in P.

The failures of an object are defined in two steps: We start with the definition of the refusals – sets of events that can be refused – for a given state. Then we combine this definition with *reachable* to define the failures of an object.

A set of events X is a refusal of a class $\mathcal{C}$ in the state st iff for all enabled values of simple parameters s_p and for all input values i_p there exists an output value o_p such that the event $(c, s_p \cup o_p \cup i_p)$ is *not* in X. The transition for $(c, s_p \cup o_p \cup i_p)$ can either be taken or there exists no output value such that a transition is defined for s_p and i_p.

$$
\begin{array}{|l}
refusal : ClassStruct \rightarrow (State \nrightarrow \mathbb{P}\,\mathbb{P}\,Event) \\
\hline
\forall \mathcal{C} : ClassStruct \bullet \Big(\operatorname{dom} refusal(\mathcal{C}) = \mathcal{C}.states \wedge \\
\quad \forall st : \mathcal{C}.states \bullet \forall X : refusal(\mathcal{C})(st) \bullet \\
\quad\quad \forall c : \mathcal{C}.channels \bullet \forall s_p : \mathcal{C}.enable(c, st) \bullet \forall i_p : \mathcal{C}.in_p(c) \bullet \\
\quad\quad\quad \exists o_p : \mathcal{C}.out_p(c) \bullet ((c, i_p \cup o_p \cup s_p) \notin X \wedge \\
\qquad\qquad\qquad\qquad \text{[The event } e = (c, i_p \cup o_p \cup s_p) \text{ cannot be refused if]} \\
\quad\quad\quad\quad (\mathcal{C}.trans(c, i_p \cup o_p \cup s_p)(st) \neq \varnothing \vee \qquad \text{[... } e \text{ can be taken next or]} \\
\quad\quad\quad\quad\quad \forall o_p : \mathcal{C}.out_p(c) \bullet \mathcal{C}.trans(c, i_p \cup o_p \cup s_p)(st) = \varnothing)) \Big) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[... the system diverges after } e.\text{]}
\end{array}
$$

E. g. taking R_1 and R_I from section 2 we have $refusal(S\text{-}PID_1)(\{used \mapsto \varnothing\}) = R_1$ and $refusal(I\text{-}PID)(\{used \mapsto \varnothing\}) = R_I$. The definitions of $\mathcal{D}$ and *refusal* even make sense if c has no parameters of a particular type [8].

The definition of the failures of a CSP-OZ class is now straight forward. The class $\mathcal{C}$ can engage in the trace tr if there exists a reachable state st for tr. The set of events X can be refused after tr if it can be refused in the state st.

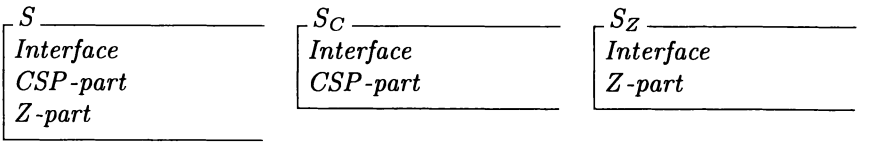
$$
\begin{array}{|l}
\mathcal{F} : ClassStruct \rightarrow \mathbb{P}(\operatorname{seq} Event \times \mathbb{P}\,Event) \\
\hline
\forall \mathcal{C} : ClassStruct \bullet \mathcal{F}(\mathcal{C}) = \\
\qquad \{(tr, X) \mid \exists st : reachable(\mathcal{C})(tr) \bullet X \in refusal(\mathcal{C})(st)\} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[no divergence]} \\
\qquad \cup \{(tr, X) : \mathcal{D}(\mathcal{C}) \times \mathbb{P}\,Event\} \quad \text{[chaotic closure in case of divergence]}
\end{array}
$$

It is proven in [8] that the definitions fulfill the healthiness conditions of the model $\mathcal{N}'$.

Note that declarations of channels with the same name in different classes must not introduce type conflicts. Otherwise the alphabet $\mathcal{A}$ in definition 1 would not be defined.

The semantics of a CSP-OZ class having only a CSP-part is the semantics of the process `main`. This is defined [17] where also the semantics of the CSP-operators can be found. The combination of CSP- and Z-syntax is simply done

by parallel composition.



Thus the semantics of S is given by $S_C \,[|\{|\, c_1, \ldots, c_n \,|\}|]\, S_Z$ where c_i are the channels declared in *Interface*.

This approach of combining (Object-)Z and CSP reuses an enormous part of existing theory. We get a lot of theorems for free. For example, the monotonicity of the CSP-parallel composition allows the separate refinement of the CSP- and the Z-part and all CSP-laws are valid laws of CSP-OZ.

6 RELATED WORK

Roscoe, Woodcock, and Wulf [19] give an informal translation from Z to CSP by separating input and output communications. The application of a Z-operation is modelled by two CSP events. CSP-OZ is much more general than this approach.

He [11], Josephs [13], and Woodcock and Morgan [25] translate state based specifications to CSP and prove various refinement results. Butler [4] combines action systems and CSP similar to our approach. Our definition of the semantics is simpler as we do not redefine CSP operators in Z; we do not separate input, output, and non-value passing communications; and we can present specifications in a better structure by using Object-Z. Also the combination of CSP and Z syntax is new and to the best of our knowledge the mixture of input and output parameters in one event has not been defined before.

LOTOS [3] combines CSP- and CCS-like operators with an algebraic specification language for abstract data types (Act One). Work on combining LOTOS and Z is in progress [2, 5]. The semantics is defined by a translation of LOTOS to ZEST (an extension of Z similar to Object-Z; also with a blocking view of operations). By contrast, we define the Object-Z semantics in the CSP model. We can apply parallel composition to classes and hide operations which is impossible in [5]. Furthermore, we do faithfully model Z-operations instead of only using blocking operations, and we precisely capture the characteristics of input and output parameters. Our integration of Z, Object-Z and CSP is much deeper than the integration of ZEST and LOTOS.

Strulo [23] investigates the difference between Z and ZEST operations. It is shown that blocking operations are good for specifying active behaviour while non blocking operations are good for modelling passive behaviour. CSP-OZ can be seen as the formalisation of the hybrid approach [23] where specifications of active and passive behaviour can be mixed. An operation `com_c` corresponds to active behaviour and an operation `effect_c` with the empty

schema as precondition corresponds to passive behaviour. We even extend the hybrid approach as the special properties of input and output parameters are not captured in [23].

The relations to the work of Smith [21] and a previous paper of the author [7] have already been stated in the introduction.

7 CONCLUSION AND FUTURE WORK

The development of CSP-OZ is driven by the idea of combining successful existing formal languages based on a well defined semantics. An important aspect is that CSP and to some extent also Object-Z and Z are proper sub languages of CSP-OZ. Hence users of any of these three languages can start using CSP-OZ without having to learn many new things. The additional features of CSP-OZ can be explored step by step. A first example of the application of this idea is the use of CSP-Z – the predecessor of CSP-OZ – as part of a Brazilian project [1], where the specification for the SACI-1 micro satellite is developed using CSP-Z.

The strength of CSP-OZ lies in distributed systems where both data and control aspects must be modelled. We tried to demonstrate this idea in section 3 with the example of a registration desk. The printer is modelled by a pure CSP-process as no internal data structure is relevant in this system. The PID manager is a mainly data driven application which we modelled without using any CSP. The combination of the different components was done using CSP-operators. Paraphrasing the title of [6], CSP-OZ is even better suited for the description of standards than Object-Z.

Another advantage of CSP-OZ is its compositional semantics. For example, the implementation of the PID manager developed in section 2 can be used in the registration desk without affecting the correctness of the design. This could be done for all components of the system. Reusing a CSP-semantics for CSP-OZ gives this important advantage for free. Thus CSP-OZ is especially well suited to build up a library of specifications together with their implementations.

The semantics of CSP-OZ might look a bit complicated at first look. We nevertheless believe that the intuition behind it is simple, and the step from (Object-)Z to CSP-OZ is fairly easy.

A manual for CSP-OZ with a detailed definition of the syntax and the semantics is under development. Refinement rules already developed in [16, 4] can easily be rephrased for CSP-OZ as done in [7, 10]. Z-data refinement is a proper refinement in CSP-OZ for channels modelling Z-operations (analogously for Object-Z data refinement). Small case studies that translate straightforwardly to CSP-OZ can be found in [21] (game of life), [7] (telecommunications protocol) and [10] (pocket calculator).

The success of a formal method crucially depends on tool support. A workbench to integrate different tools (FDR [9], Z and CSP encodings in Isabelle

[14, 24], an hierarchical editor for Z and a graphical editor for CSP) is under development in the project UniForM [15].

ACKNOWLEDGEMENTS.

I thank G. Smith for spotting my interests towards Object-Z. His cooperation and helpful comments are gratefully acknowledged. The Semantik-group in Oldenburg and the cooperation with the University of Bremen in the UniForM project provide a stimulating background for my work. G. Smith, H. Dierks, S. Kleuker, C. Dietz, S. Hallerstede, H. Fleischhack and E.-R. Olderog read draft versions of this paper and made valuable comments.

REFERENCES

[1] E. Barros and A. Sampaio. Towards provably correct hardware/software codesign using OCCAM. In *Proccedings of the Third International Workshop on Codesign*. IEEE Computer Society Press, 1994.

[2] E. Boiten, H. Bowman, J. Derrick, and M. Steen. Viewpoint consistency in Z and LOTOS: A case study. submitted for publication, 1997.

[3] T. Bolognesi, J. van de Lagemaat, and C. Vissers, editors. *LOTOSphere: Software Development with LOTOS*. Kluwer Academic, 1995.

[4] M. J. Butler. *A CSP Approach To Action Systems*. PhD thesis, University of Oxford, 1992.

[5] J. Derrick, E.A. Boiten, H. Bowman, and M.W.A. Steen. Supporting ODP – translating LOTOS to Z. In *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*. Chapmann & Hall, 1996.

[6] R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. *Computer Standards and Interfaces*, 17:511–533, 1995.

[7] C. Fischer. Combining CSP and Z. submitted for publication, 1997.

[8] C. Fischer. Combining Object-Z and CSP. Technical report, University of Oldenburg, April 1997.

[9] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement: FDR 2*, Dec 1995. Preliminary Manual.

[10] S. Hallerstede. Die semantische Fundierung von CSP-Z. Master's thesis, University of Oldenburg, January 1997. in german.

[11] J. He. Process simulation and refinement. *Formal Aspects of Computing*, 1(3):229–241, 1989.

[12] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[13] M.B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.

[14] Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle/HOL. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics*, LNCS 1125, pages 283–298. Springer Verlag, 1996.

[15] B. Krieg-Brückner, J. Peleska, E.-R. Olderog, D. Balzer, and A. Baer. UniForM — Universal Formal Methods Workbench. In U. Grote and G. Wolf, editors, *Statusseminar des BMBF Softwaretechnologie*, pages 357–378. BMBF, Berlin, March 1996.

[16] C. Morgan. *Programming from Specifications.* Prentice Hall, 1990.

[17] A. W. Roscoe. An alternative order for the failures model. In *Two papers on CSP*, Technical Monograph PRG-67, pages 1–26. Oxford University, 1988.

[18] A. W. Roscoe and G. Barrett. Unbounded nondeterminism in CSP. In *Mathematical Foundations of Programming Semantics*, volume 442 of *LNCS*, pages 160–193. Springer-Verlag, 1989.

[19] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. Non-interference through determinism. In D. Gollmann, editor, *ESORICS 94*, volume 875 of *LNCS*, pages 33–54. Springer-Verlag, 1994.

[20] G. Smith. *An Object-Oriented Approach to Formal Specification.* PhD thesis, Department of Computer Science, University of Queensland, St. Lucia 4072, Australia, October 1992.

[21] G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. submitted for publication, 1997.

[22] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall International Series in Computer Science, 2nd edition, 1992.

[23] B. Strulo. How firing conditions help inheritance. In J. P. Bowen and M. G. Hinchey, editors, *ZUM '95: The Z Formal Specification Notation*, volume 967 of *LNCS*, pages 264–275, 1995.

[24] H. Tej and B. Wolff. A corrected failure-divergence-model for CSP in Isabelle/HOL. submitted for publication, 1997.

[25] J. C. P. Woodcock and C. C. Morgan. Refinement of state-based concurrent systems. In *Proceedings of VDM Symposium 1990*, volume 428 of *LNCS*, pages 340–351. Springer-Verlag, 1990.

BIOGRAPHY

Clemens Fischer received a master degree in computer science from the University Oldenburg, Germany, in 1995. Since then he works in the research group headed by Prof. E.-R. Olderog at the same university.

His current research interests are combining FDTs, practical applications of formal methods and integration of fully automatic verification techniques with a transformational approach to the design of correct systems.