Distributed System Development in B

M. Butler[*] M. Waldén[†]

Abstract

The B-Method is a method for the stepwise derivation of sequential programs. In this paper we show how the B-Method can be used for designing distributed systems by embedding action systems within this method. The action system formalism is designed for the construction of parallel and distributed systems in a stepwise manner within the refinement calculus. We describe how action systems are written in B AMN. We also show the correspondence between refinement rules for action systems and the proof obligations generated in the B-Method. Furthermore, we propose an extension of the B-Method to cover parallel and distributed systems. Familiarity with B AMN is assumed.

1 Introduction

In the B-Method [**?**, **?**], a system is specified as an abstract machine consisting of some state and some operations acting on that state. This is essentially the same structure as an action system [**?**] has, which describes the behaviour of a parallel reactive system in terms of the atomic actions (i.e., operations). The operations of both B machines and action systems are described using notations based on Dijkstra's guarded command language [**?**]. Action systems are used to construct parallel and distributed systems in a stepwise manner as described by Back et al. [**?**, **?**]. Stepwise refinement of action systems is formalized within the refinement calculus [**?**] based on the weakest precondition calculus of Dijkstra. Also B machines may be refined in a stepwise manner relying on an extension of this weakest precondition calculus.

Different views as to what constitutes the observable behaviour of an action system may be taken. We have here considered a state-based view of action systems. An event-based view is studied in [**?**]. In the state-based view action systems have a local and a global state. The environment interacts with an action system via its global state. It is, thus, only the global state that is visible to and accessible to the environment.

In this paper we show how refinement of action systems and, hence, distributed systems can be performed using the B-Method. We show how to embed action systems in the B-Method. We compare the proof obligations of action system refinement and refinement within the B-Method and, furthermore, suggest how to extend the B-Method in order to be able to prove these refinements. Using a mechanical tool as the B-Toolkit [**?**], which supports the B-Method, facilitates the verification of the refinement steps and increases the confidence in the proofs. A different approach using the B-Method and the B-Toolkit for proving action system refinement is taken in [**?**].

We will first briefly describe action systems and their embedding in the B-Method in section 2. In section 3, we compare the refinement of action systems with refinement in B. We

[*]Univ. of Southampton, Dept. of Electronics and Computer Science, Highfield, Southampton SO17 1BJ, United Kingdom, e-mail: `M.J.Butler@ecs.soton.ac.uk`

[†]Åbo Akademi University, Department of Computer Science, FIN-20520 Turku, Finland, e-mail: `Marina.Walden@abo.fi`

discuss extensions to the B-Method in section 4. Finally, in section 5 we show how parallel decomposition can be interpreted in B. We conclude in section 6.

2 Action Systems within B

We first consider the action system framework. We only give a very brief introduction here. More on these topics and further references can be found elsewhere [?, ?, ?].

2.1 Action Systems

An *action system* $\mathcal{A}$ is a statement of the form

$$\mathcal{A} \hat{=} |[\ \ \mathbf{var}\ x; x := x_0; \mathbf{do}\ A_1\ [\!]\ \ldots [\!]\ A_m\ \mathbf{od}\ \ ]| : z$$

on the *state variables* x and z, where the variables z are the *global* variables and x are the *local* variables. Each variable is associated with some domain of values. The set of possible assignments of values to the state variables constitutes the state space. The initialisation statement $x := x_0$ assigns initial values to the local variables x. The global variables z are used for interaction with the environment.

Each action A_i is of the form $g_i \rightarrow S_i$ where the *guard* g_i is a boolean expression on the state variables and the *body* S_i is a statement on the state variables. We denote the guard g_i of A_i by gA_i and the body S_i by sA_i. Furthemore, we say that an action is enabled in a state when its guard evaluates to *true* in that state.

The behavior of an action system is that of Dijkstra's guarded iteration statement [?] on the state variables: the initialisation statement is executed first, thereafter, as long as there are enabled actions, one action at a time is nondeterministically chosen and executed. When all the actions are disabled the action system terminates.

If two actions are independent, i.e., they do not have any variables in common, they can be executed in parallel. Their parallel execution is then equivalent to executing the actions one after the other, in either order.

2.2 Action Systems as Abstract Machines

An action will be specified as a guarded statement in the generalised substitution notation of B AMN. The semantics of generalised substitutions is defined using weakest-precondition formulae [?]; for statement S and postcondition P, the formula $[S]P$ characterises those initial states from which S is guaranteed to terminate in a state satisfying P. In AMN the guarded substitution $G \Rightarrow S$ has the weakest precondition,

$$[G \Rightarrow S]P \quad \hat{=} \quad G \Rightarrow [S]P.$$

Since the weakest precondition of an action $G \rightarrow S$ is the same, actions can be specified as guarded substitutions in B AMN.

The syntax for a guarded substitution interpreted as an operation in the B-Method is:

$$\mathit{Operation\_name} \quad \hat{=} \quad \text{PRE}\ \mathit{true}\ \text{THEN (SELECT}\ G\ \text{THEN}\ S\ \text{END) END;}$$

where the precondition has the value *true* and can be left out. The action $G \rightarrow S$ interpreted as an operation is then

$$\text{SELECT}\ G\ \text{THEN}\ S\ \text{END.}$$

$$
\begin{array}{lll}
\mathcal{A} \mathrel{\hat{=}} & |[& \textbf{var } x; \\
 & & x := x_0; \\
 & & \textbf{do} \\
 & & \quad [\!] \; gA_1 \rightarrow sA_1 \\
 & & \quad [\!] \; gA_2 \rightarrow sA_2 \\
 & & \quad \vdots \\
 & & \quad [\!] \; gA_m \rightarrow sA_m \\
 & & \textbf{od} \\
 &]| : z &
\end{array}
$$

MACHINE $\mathcal{A}$
INCLUDES
  Global_z
VARIABLES
  x
INVARIANT
  $Q(x, z)$
INITIALISATION
  $x := x_0$
OPERATIONS
  $A_1 \mathrel{\hat{=}}$ SELECT gA_1 THEN sA_1 END;
  $A_2 \mathrel{\hat{=}}$ SELECT gA_2 THEN sA_2 END;
  $\vdots$
  $A_m \mathrel{\hat{=}}$ SELECT gA_m THEN sA_m END
END

Figure 1: An action system and its embedding in an abstract machine.

We can now write the action system $\mathcal{A}$ as an Abstract Machine specification with the data invariant $Q(x, z)$ as given in Figure **??**. The translation of action systems into AMN is straightforward.

Since the global variables should be available to more than one action system, we need to treat them differently from the local variables. The global variable z of the action system $\mathcal{A}$ is included as a machine, *Global_z*, in the abstract machine specification $\mathcal{A}$. A separate machine should exist for each global variable. An action system can then include exactly those global variables it refers to. In the machine *Global_z*, shown in Figure **??**, we declare the global variable z and give its properties in the *invariant* clause. Furthermore, the variable z is assigned via an operation $assign\_z(y)$, where the value y to be assigned to z is given as the parameter. All assignments to z in an action A_i in the action system $\mathcal{A}$ are replaced by calls to $assign\_z$ in the corresponding operations in the machine specification $\mathcal{A}$.

MACHINE *Global_z*
VARIABLES
  z
INVARIANT
  $P(z)$
INITIALISATION
  $z := z_0$
OPERATIONS
  $assign\_z(y) \mathrel{\hat{=}}$ PRE $P(y)$ THEN $z := y$ END
END

Figure 2: Declaration of a global variable z in B.

3 Refining Action Systems within B

Specification machines usually contain abstract data structures that are not directly implementable in a programming language. *Data refinement* is used in order to bring abstract specifications towards implementations by replacing the local variables of the abstract machine with concrete variables that are more easily implemented. The refinement relation within the B-Method is transitive and monotonic.

3.1 Data Refinement of Actions

An abstraction invariant $R(x, x', z)$ relating the abstract variables x and the concrete variables x', as well as the global variables z, is used to replace abstract statements with concrete statements. If S is a statement on the variables x, z, S' is a statement on the variables x', z, and $R(x, x', z)$ is the abstraction invariant, then we write

$$S \sqsubseteq_R S'$$

for "S is data-refined by S' under abstraction invariant R".

The weakest-precondition definition of data refinement as described in [?, ?, ?] is as follows:

Definition 1 (Data Refinement in WP) *$S \sqsubseteq_R S'$ if for each postcondition P independent of the concrete variables x',*

$$R \wedge [S]P \quad \Rightarrow \quad [S'](\exists x.\ R \wedge P).$$

The definition of data refinement in B AMN [?] is given below. Let $pre(S) \ \hat{=}\ [S]true$, i.e., $pre(S)$ represents the condition under which S is guaranteed to terminate.

Definition 2 (Data Refinement in B) *S is data-refined by S' under abstraction invariant R, if*

$$\begin{array}{rcl} I \wedge R \wedge pre(S) & \Rightarrow & pre(S') \\ I \wedge R \wedge pre(S) & \Rightarrow & [S']\neg([S]\neg R), \end{array}$$

where I is an invariant of S, i.e., $I \wedge pre(S) \Rightarrow [S]I$.

The second condition states that an operation S' establishes a situation where the old operation S cannot fail to maintain R.

It can be shown that Definition ?? implies Definition ??. In particular, we have the following theorem:

Theorem 1 *$S \sqsubseteq_{R \wedge I} S'$ if the following conditions hold:*

$$\begin{array}{rcl} I \wedge pre(S) & \Rightarrow & [S]I \\ I \wedge R \wedge pre(S) & \Rightarrow & pre(S') \\ I \wedge R \wedge pre(S) & \Rightarrow & [S']\neg([S]\neg R). \end{array}$$

Obligations such as these can be checked using the theorem-proving environments associated with the B-method [?, ?]. Waldén and Sere [?] show the correspondence between the data refinement of actions in WP and B in more detail.

3.2 Data Refinement of Initialisations

The expression $S \sqsubseteq_R S'$ means that S' data-refines S in such a way that the abstraction invariant R is preserved from initial to final states. However, no assumption is made about the state of action systems before initialisation. This means that a concrete initialisation must refine an abstract initialisation in such a way that it establishes the abstraction invariant rather than simply preserving it. We write $S \sqsubseteq_R^{init} S'$ to deal with this, where

Definition 3 (Data Refinement of initialisations in WP) $S \sqsubseteq_R^{init} S'$ *if for each postcondition P independent of the concrete variable x',*

$$[S]P \quad \Rightarrow \quad [S'](\exists x.\ R \wedge P).$$

In B, the condition for data refinement of initialisations is as follows:

Definition 4 (Data Refinement of initialisations in B) *S is initially-data-refined by S' under R if*

$$[S'](\neg[S]\neg R).$$

Again we have the following theorem:

Theorem 2 $S \sqsubseteq_{R \wedge I}^{init} S'$ *if the following conditions hold:*

$$[S]I$$
$$[S']\neg([S]\neg R).$$

3.3 Simulation

A *simulation* is a relation between action systems $\mathcal{A}$ and $\mathcal{A}'$ with corresponding actions but possibly different state-spaces. Let the abstract action system $\mathcal{A}$ be a statement on the variables x and z, and the refined action system $\mathcal{A}'$ a statement on x' and z. The variables x are the abstract local variables that are data refined into the concrete local variables x'. They are invisible to the environment. The global variables z, on the other hand, form the interface to the environment and are left unchanged.

In Figure **??** the abstract action system $\mathcal{A}$ and its refinement $\mathcal{A}'$ are shown as abstract machines. The machines for the global variables are as before in Figure **??**. In the refinement machine $\mathcal{A}'$ we have renamed the global variable z to z' due to restrictions in the B-Method and include the modified machine *Global_z'*. However, in the *invariant* clause we state that the global variables z and z' really are the same, $z = z'$. Let us now study the simulation rule:

Definition 5 (Simulation) *For the abstract action system $\mathcal{A}$ and the concrete action system $\mathcal{A}'$ in Figure **??**, $\mathcal{A}$ is simulated by $\mathcal{A}'$ with abstraction invariant $R(x, x', z)$, denoted$\mathcal{A} \sqsubseteq_R \mathcal{A}'$, provided each of the conditions below hold. Let Init denote the initialisation $x := x_0$ of the action system $\mathcal{A}$, and Init' the initialisation $x' := x_0'$ of $\mathcal{A}'$.*

1. $Init \ \sqsubseteq_R^{init} \ Init'$
2. $A_i \ \sqsubseteq_R \ A_i', \qquad \text{for } i = 1, \ldots, m$
3. $R \wedge gA_i \ \Rightarrow \ gA_i', \qquad \text{for } i = 1, \ldots, m.$

MACHINE $\mathcal{A}$	REFINEMENT $\mathcal{A}'$
	REFINES
	$\mathcal{A}$
INCLUDES	INCLUDES
$Global\_z$	$Global\_z'$
VARIABLES	VARIABLES
x	x'
INVARIANT	INVARIANT
$Q(x, z)$	$R(x, x', z) \wedge z = z'$
INITIALISATION	INITIALISATION
$x := x_0$	$x' := x'_0$
OPERATIONS	OPERATIONS
$A_1 \ \hat{=}\ $ SELECT gA_1 THEN sA_1 END;	$A_1 \ \hat{=}\ $ SELECT gA'_1 THEN sA'_1 END;
$A_2 \ \hat{=}\ $ SELECT gA_2 THEN sA_2 END;	$A_2 \ \hat{=}\ $ SELECT gA'_2 THEN sA'_2 END;
$\vdots$	$\vdots$
$A_m \ \hat{=}\ $ SELECT gA_m THEN sA_m END	$A_m \ \hat{=}\ $ SELECT gA'_m THEN sA'_m END
END	END

Figure 3: An abstract action system $\mathcal{A}$ and its simulation $\mathcal{A}'$ in B AMN.

Conditions 1 and 2 ensure that the initialisation and each action of $\mathcal{A}'$ is a refinement of its counterpart in $\mathcal{A}$, and are referred to as data-refinement conditions. These are precisely the conditions that define refinement of machines in B AMN [**?**]. Condition 3 ensures that an action in $\mathcal{A}'$ is only enabled if the corresponding action in $\mathcal{A}$ is enabled, and is referred to as a progress condition. Intuitively, $\mathcal{A} \sqsubseteq_R \mathcal{A}'$ means that any observable behaviour of $\mathcal{A}'$ is also an observable behaviour of $\mathcal{A}$.

We can note that this simulation implies state-trace refinement [**?**].

4 Introducing Internal Actions into B

In the B-Method the machines N and M, where N refines M, must have identical operation signatures. This means that the corresponding operations in N and M must have identical names and, if the operations have parameters, these must also be identical. The machines N and M need not contain the same variables though.

In a refinement of action systems we could, however, introduce some new internal actions. Internal actions do not affect the global variables. Hence, they are not visible to the environment of a machine and are, thus, outside the control of the environment. Any number of executions of an internal action may occur in between each execution of a visible action. If the action system reaches a state where internal actions can be executed infinitely, then the action system diverges.

4.1 Refinement with Internal Actions

To ensure that the internal actions do not introduce divergence, a well-foundedness argument is used. A set *WF*, with irreflexive partial order $<$, is *well-founded* if each non-empty subset of *WF* contains a minimal element under $<$. For example, the natural numbers with the usual

MACHINE $\mathcal{A}$

INCLUDES
$Global\_z$
VARIABLES
x
INVARIANT
$Q(x, z)$

INITIALISATION
$x := x_0$
OPERATIONS
$A_1 \hat{=}$ SELECT gA_1 THEN sA_1 END;
$\vdots$
$A_m \hat{=}$ SELECT gA_m THEN sA_m END

END

REFINEMENT $\mathcal{A}'$
REFINES
$\mathcal{A}$
INCLUDES
$Global\_z'$
VARIABLES
x'
INVARIANT
$R(x, x', z) \wedge z = z'$
VARIANT
E
INITIALISATION
$x' := x'_0$
OPERATIONS
$A_1 \hat{=}$ SELECT gA'_1 THEN sA'_1 END;
$\vdots$
$A_m \hat{=}$ SELECT gA'_m THEN sA'_m END
INTERNAL OPERATIONS
$B_1 \hat{=}$ SELECT gB_1 THEN sB_1 END;
$\vdots$
$B_n \hat{=}$ SELECT gB_n THEN sB_n END
END

Figure 4: An action system $\mathcal{A}$ and its simulation $\mathcal{A}'$ with internal actions.

ordering, or the cartesian product of two or more well-founded sets with lexicographic ordering, all form well-founded sets. The well-foundedness argument requires the use of a well-founded set *WF* and a *variant*, which is an expression in the state-variables. The variant should always be an element of *WF*, and it should be decreased by each internal action of the concrete system.

Let us consider the action system $\mathcal{A}$ and its refinement $\mathcal{A}'$ with internal actions given in Figure **??**. The refinement relation between $\mathcal{A}$ and $\mathcal{A}'$ is $R(x, x', z)$. Again the global variable z should remain unchanged in the refinement, $z = z'$. We have given the internal actions B_j and their variant E in the *internal-operations* clause and the *variant* clause, respectively. These two clauses are suggested as new clauses for the refinement with internal actions. The *internal operations* should be of the same form as the operations in the *operations* clause, while the *variant* should be an expression.

The simulation rule is now as follows:

Definition 6 (Simulation with internal actions) *Let $\mathcal{A}$ and $\mathcal{A}'$ be the action systems in Figure* **??**. *$\mathcal{A}$ is simulated by $\mathcal{A}'$ with the abstraction invariant $R(x, x', z)$, the well-founded set* WF*, and the variant E, denoted $\mathcal{A} \sqsubseteq_{(R,WF,E)} \mathcal{A}'$, provided each of the following conditions hold. Let again Init denote the initialisation $x := x_0$ of the action system $\mathcal{A}$, and Init$'$ the initialisation $x' := x'_0$ of $\mathcal{A}'$.*

1. *$Init \sqsubseteq_R^{init} Init'$*
2. *$A_i \sqsubseteq_R A'_i$,* for $i = 1, \ldots, m$

3. $skip \sqsubseteq_R B_j$, for $j = 1, \ldots, n$

4. $R \wedge gA_i \Rightarrow gA'_i \vee (\exists j.\ 1 \leq j \leq n.\ gB_j)$, for $i = 1, \ldots, m$

5. $R \Rightarrow E \in WF$

6. $R \wedge E = e \Rightarrow [B_j](E < e)$, for $j = 1, \ldots, n$

Conditions 1, 2, and 3 are data-refinement conditions. Conditions 1 and 2 are the same as in Definition **??**, while Condition 3 ensures that each internal action of Q causes no change to the corresponding abstract state. Condition 4 is a progress condition and ensures that, whenever an action of $\mathcal{A}$ is enabled, either the corresponding action of $\mathcal{A}'$ is enabled, or then some internal action of $\mathcal{A}'$ is enabled. Conditions 5 and 6 are referred to as non-divergence conditions. Condition 5 ensures that the variant E is an element of the set *WF*, while Condition 6 ensures that the internal actions of $\mathcal{A}'$ always decrease E when executed. Together, Conditions 5 and 6 ensure that the internal actions of $\mathcal{A}'$ are eventually disabled and so cannot introduce divergence.

Again, simulation with internal actions ensures state-trace refinement [**?**].

4.2 Proof Obligations for Internal Actions

The refinement with internal actions cannot be completely proved within the B-Method. In order to achieve a mechanical support within the B-Method for proving the conditions in Definition **??**, additional proof obligations would need to be generated for a refinement machine. Conditions 1 and 2 in Definition **??** are, however, directly generated within the B-Method from the initialisation and the operations. The rest of the conditions, Conditions 3 - 6, could be generated as proposed below considering the *internal-operations* clause.

The internal operations refine the operation *skip*. A proof obligation within B corresponding to Condition 3 can be generated by assuming that for each internal operation B_j in the refined machine, there is a corresponding *skip*-operations in the machine to be refined.

A proof obligation for the progress condition, Condition 4, could be generated from the guards gA_i and gA'_i in each operation A_i in $\mathcal{A}$ and $\mathcal{A}'$. Also the disjunction of the guards of the internal operations, $(gB_1 \vee \ldots \vee gB_n)$, could be extracted in a straightforward manner for this proof obligation. The guards are easily extracted from the operations, since an operation here is always of the form:

$$\text{SELECT } < guard > \text{ THEN } < body > \text{ END.}$$

Finally, to be able to generate the proof obligations for the non-divergence conditions 5 and 6, we also need to consider the *variant* clause in the machine refinement. Furthermore, if we want to use another well founded set than the natural numbers, which is the well founded set the B-Method uses, we would need to introduce a *well-founded-set* clause to define this set. In the B-Method the proof obligations generated for the loop:

$$\text{WHILE } G \text{ DO } S \text{ INVARIANT } R \text{ VARIANT } E \text{ END}$$

in an implementation machine are:

$$\begin{array}{ll} \text{(i)} & R \Rightarrow E \in \mathbf{N}, \\ \text{(ii)} & (\forall l.\ (R \wedge G) \Rightarrow [n := E]([S](E < n))), \end{array}$$

where l denotes the variables modified within the loop. According to these, (i) the variant E should be an expression yielding a natural number. Furthermore, (ii) when the guard G of the loop holds, the body S should decrease the variant E. These proof obligations could also be generated using the *variant-* and *internal-operations*-clauses in the refinement machine $\mathcal{A}'$ in Figure **??**. The loop condition G would then be the disjunction of the guards of all the internal operations and S would denote an internal operation. A separate proof obligation corresponding to (ii) should be generated for each internal operation.

By generating these additional proof obligations the refinement of action systems by introducing internal actions could be proved within the B-Method.

With the above suggested additions to the proof obligations in the B-Method even *superposition refinement* of action systems could be proved. Superposition refinement [?, ?, ?] is a powerful program modularization and structuring method, which is often used for refining action systems. In superposition some new functionality is added to an algorithm in the form of additional variables and assignments to these, while the original computation is preserved. Superposition is really a special case of the refinement with internal actions, where new variables are added to the action system and the old ones remain unchanged. The superposition method has been used to derive and analyse distributed algorithms in various ways [?, ?, ?, ?]. In [?] an example of superposition refinement of action systems using the B-Method is shown. In that paper a different approach is taken to generate the proof obligations in Definition **??**. There extra constructs are created in order for the B-Method to generate the right proof obligations within the current B notation.

5 Parallel Composition

Action systems can be composed into parallel systems [?]. The parallel composition of the action systems $\mathcal{A}$ and $\mathcal{B}$ is written $\mathcal{A} \mid\mid \mathcal{B}$. This composition is formed by merging the variables and actions of the subsystems $\mathcal{A}$ and $\mathcal{B}$. The local state variables of the subsystems have to be distinct. This can, however, easily be achieved by renaming before forming the composition.

Let us now consider the action systems $\mathcal{A}$ and $\mathcal{B}$ in B AMN in Figure **??**, where $x \cap y = \emptyset$. The global variables are again defined as in Figure **??**. The parallel composition $\mathcal{A} \mid\mid \mathcal{B}$ of $\mathcal{A}$ and $\mathcal{B}$ is then defined as the abstract machine in Figure **??**. The common global variables z of $\mathcal{A}$ and $\mathcal{B}$ will also be global variables of the parallel composition $\mathcal{A} \mid\mid \mathcal{B}$. For the rest, the variables and the initialisation, as well as the operations in $\mathcal{A}$ and $\mathcal{B}$ are simply merged to form $\mathcal{A} \mid\mid \mathcal{B}$.

The invariant of $\mathcal{A} \mid\mid \mathcal{B}$ is a conjunction of the invariants of the subsystems $\mathcal{A}$ and $\mathcal{B}$. This imposes, however, an extra requirement on the operations of the subsystems. The operations A_i should preserve the invariant Q, while the operations B_j should preserve invariant P. This is mainly a restriction on the assignments to the common global variables z in the operations A_i and B_j.

The parallel composition of action systems is monotonic. Thus, if action system $\mathcal{A}$ is refined by $\mathcal{A}'$ and action system $\mathcal{B}$ is refined by $\mathcal{B}'$, then $\mathcal{A} \mid\mid \mathcal{B}$ is refined by $\mathcal{A}' \mid\mid \mathcal{B}'$. This means that the subsystems in a parallel composition may be refined independently.

6 Conclusion

We have described how action systems can be transformed into abstract machines within the B-Method. Furthermore, we have compared the refinement notions of action systems and B. We have also suggested extensions to the refinement machines and the proof obligations generated from them for refinement with internal actions. With the proposed extensions we would even be able to prove superposition refinements within the B-Method. Since superposition refinement as

MACHINE $\mathcal{A}$ **INCLUDES** $Global\_u,\ Global\_z$ **VARIABLES** x **INVARIANT** $P(x,u,z)$ **INITIALISATION** $x := x_0$ **OPERATIONS** $A_1 \;\hat{=}\;$ SELECT gA_1 THEN sA_1 END; $\vdots$ $A_m \;\hat{=}\;$ SELECT gA_m THEN sA_m END **END**	**MACHINE** $\mathcal{B}$ **INCLUDES** $Global\_v,\ Global\_z$ **VARIABLES** y **INVARIANT** $Q(y,v,z)$ **INITIALISATION** $y := y_0$ **OPERATIONS** $B_1 \;\hat{=}\;$ SELECT gB_1 THEN sB_1 END; $\vdots$ $B_n \;\hat{=}\;$ SELECT gB_n THEN sB_n END **END**

Figure 5: The action systems $\mathcal{A}$ and $\mathcal{B}$ in B AMN.

well as action systems refinement in general is used for deriving parallel and distributed systems, we would thus be able to derive parallel and distributed systems within the B-Method. Action systems $\mathcal{A}$ and $\mathcal{B}$ can be composed into a parallel system $\mathcal{A} \,||\, \mathcal{B}$ by combining the state variables and actions of $\mathcal{A}$ and $\mathcal{B}$, respectively.

Abrial has proposed an approach to the design of protocols using the B-Method [?]. With this approach, a protocol is specified as a single operation which is subsequently decomposed into a sequence of steps through a series of refinements. The introduction of each new step in the protocol is justified by showing that it is a data refinement of the *skip* action. This is the same as our data-refinement condition on internal actions being introduced by a simulation step (Definition **??**). Lano and Dick on the other hand have developed an approach to dealing with concurrency in B by combining it with a form of temporal logic [?].

We have seen the close correspondence between action systems and the abstract machines of B and seen the similarity between their notions of refinement. Because of this close correspondence, we are able to apply action system techniques such as internalisation of actions and parallel composition to abstract machines. These techniques provide a powerful abstraction mechanism, since they allow us to abstract away from the distrib uted architecture of a system and the complex interactions between its subsystems. The reasoning required to use these techniques involves refinement arguments and variant arguments, which is the sort of reasoning already used in B. The techniques are also very modular since the parallel components of a distributed system can be refined separately without making any assumptions about the rest of the system.

By using the B-Method and the B-Toolkit we gain some extra features. The tool can for example assist in finding logical errors in the distributed systems, which is very useful because of the complexity of distributed systems. Furthermore, the B-Method is a well known method and is succesfully used in many industrial projects applying formal methods.

There are, however, also drawbacks of using the B-Method for deriving action systems. Without the B-Method extensions suggested in this paper extra operations and machines would need to be introduced for the derivation [?] in order to prove the refinement step with internal actions in Definition **??**. The more complex action system we have the more complex these

MACHINE $\mathcal{A} \,||\, \mathcal{B}$
INCLUDES
$Global\_u,\ Global\_v,\ Global\_z$
VARIABLES
x, y
INVARIANT
$P(x,u,z) \wedge Q(y,v,z)$
INITIALISATION
$x, y := x_0, y_0$
OPERATIONS
$A_1 \;\hat{=}\;$ SELECT gA_1 THEN sA_1 END;
$\vdots$
$A_m \;\hat{=}\;$ SELECT gA_m THEN sA_m END;
$B_1 \;\hat{=}\;$ SELECT gB_1 THEN sB_1 END;
$\vdots$
$B_n \;\hat{=}\;$ SELECT gB_n THEN sB_n END
END

Figure 6: The parallel composition of action systems $\mathcal{A}$ and $\mathcal{B}$.

extra conditions will be.

Acknowledgements

Marina Waldén is funded by the Cocos-project at Åbo Akademi University. The project is supported by the Academy of Finland.

References

[1] J.-R. Abrial. The B Method for large software: Specification, design and coding. In *Proc. of VDM'91 Formal Software Development Methods, Vol. 2.* LNCS 552, Springer-Verlag, pages 398 – 405, Noordwijkerhout, the Netherlands, October 1991.

[2] J.-R. Abrial. *Cryptographic Protocol Specification and Design.* March 1995.

[3] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[4] R. J. R. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications.* Tract 131, Mathematisch Centrum, Amsterdam, 1980.

[5] R. J. R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *Proc. of the 2nd ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing.* ACM, pages 131–142, Montreal, Quebec, Canada, August 1983.

[6] R. J. R. Back and K. Sere. Deriving an occam implementation of action systems. In *Proc. of the 3rd Refinement Workshop BCS FACS / IBM UK Laboratories / Oxford University*, Hursley Park, England, January 1990. *Workshops in Computing*, Springer–Verlag, 1991.

[7] R. J. R. Back and K. Sere. From action systems to modular systems. In *Proc. of Formal Methods Europe (FME'94): Industrial Benefit of Formal Methods.* LNCS 873, pp. 1 – 25, Barcelona, Spain, October 1994.

[8] R. J. R. Back and K. Sere. Superposition refinement of reactive systems. *Formal Aspects of Computing*, 8(3):324–346, 1996.

[9] R. J. R. Back and J. von Wright. Trace refinement of action systems. In *Proc. of CONCUR'94.* LNCS 836, pages 367 – 384, Uppsala, Sweden, August 1994. Springer–Verlag.

[10] M. J. Butler. *An approach to the design of distributed systems with B AMN.* University of Southampton, Declarative Systems & Software Engineering Technical Reports, DSSE-TR-96-6, September 1996.

[11] E. W. Dijkstra. *A Discipline of Programming.* Prentice–Hall International, Englewood Cliffs, New Jersey, 1976.

[12] S. M. Katz. A superimposition control construct for distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(2):337–356, April 1993.

[13] K. Lano and J. Dick. Development of concurrent systems in B AMN. In He Jifeng, editor, *7th BCS-FACS Refinement Workshop*, Workshops in Computing. Springer–Verlag, 1996.

[14] C. C. Morgan and T. Vickers, editors. *On the Refinement Calculus.* Formal Approaches to Computing and Information Technology. Springer, 1992.

[15] J. M. Morris. Laws of data refinement. *Acta Informatica*, 26:287–308, 1989.

[16] D. S. Neilson and I. H. Sorensen. *The B-Technologies: A system for computer aided programming.* Including the *B-Toolkit User's Manual, Release 3.2.* B-Core (UK) Ltd., Oxford, U.K., 1996.

[17] K. Sere. and M. Waldén *Verification of a distributed algorithm due to Chu.* Series A–156, Reports on Computer Science and Mathematics, Åbo Akademi University, Finland, 1994. Abstract appeared in *Proc. of the 13th Annual Symposium on the Principles of Distributed Computing*, page 391, Los Angeles, USA, August 1994.

[18] M. Waldén *Formal derivation of a distributed load balancing algorithm.* Series A–172, Reports on Computer Science and Mathematics, Åbo Akademi University, Finland, 1995.

[19] M. Waldén and K. Sere. Refining action systems within B-Tool. In *Proc. of Formal Methods Europe (FME'96): Symposium on Industrial Benefit and Advances in Formal Methods.* LNCS 1051, pages 84 – 103, Oxford, England, March 1996. Springer–Verlag.