New Concepts in the Abstract Format of the Compositional Interchange Format

D.A. van Beek * P. Collins † D.E. Nadales * J.E. Rooda *
R.R.H. Schiffelers *

* Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands
{d.a.v.beek, d.e.nadales.agut, j.e.rooda, r.r.h.schiffelers}@tue.nl
† Centrum Wiskunde en Informatica
Postbus 94079, 1090 GB Amsterdam, The Netherlands
pieter.collins@cwi.nl

Abstract: The compositional interchange format for hybrid systems (CIF) supports inter-operability of a wide range of tools by means of model transformations to and from the CIF. Work on the CIF takes place in the FP7 Multiform project, and in several other European projects. The CIF consists of an abstract and a concrete format, used for defining a formal semantics and for modeling, respectively. This paper discusses the results of a redesign of the abstract format as previously published, leading to the following main changes: variables are introduced using scoping operators; the abstract language is made more orthogonal by providing an operator for each concept in the language; parallel composition has been defined in such a way that compositional verification (assume/guarantee reasoning) is supported; and the concept of urgent actions has been properly defined. As a result, the expressivity and semantics of the abstract language have been considerably improved.

Keywords: Modeling, Automata, Hybrid Systems, Formal Semantics.

1. INTRODUCTION

The main purpose of the Compositional Interchange Format (CIF), that has originally been developed in HYCON, see HYCON Network of Excellence (2005) and Beek et al. (2007b,a, 2008b), is to establish inter-operability of a wide range of tools by means of model transformations to and from the CIF. In addition, the CIF provides a generic modeling formalism and tools for a wide range of untimed, timed and hybrid systems. Fig. 1 gives an overview of work on the CIF in different projects. In the EU FP7 project Multiform, see MULTIFORM consortium (2008), bidirectional transformations between the CIF and several languages/tools are developed. In the EU FP7 project C4C, see C4C consortium (2008), work on the CIF is mainly on compositional verification, whereas in the EU ITEA2 Twins project, see ITEA Twins consortium (2009), the CIF is connected to tools for supervisory control synthesis. The CIF is used in several industrial projects. Some examples are presented in italics in Fig. 1. Not represented in Fig. 1 is work in the EU FP7 project DISC, see DISC consortium (2009), on possible connections between the CIF and Petri nets. For an overview on previous related work on interchange formalisms, such as found in MoBIES team (2002), Pinto et al. (2006), Cairano et al. (2006), we refer to Beek et al. (2007b,a).

2. REVISED SYNTAX AND INFORMAL SEMANTICS

2.1 Syntax

An atomic interchange automaton $\alpha_{\text{atomic}}$ is a tuple $\alpha_{\text{atomic}} = (V, v_0, \text{flow, inv, tcp, } E)$,
where $V$ is a set of locations (vertices); $v_0 \in V$ is the initial location, also called active location; flow, inv, tcp are functions of the set $V \rightarrow P(\mathcal{V})$, which associate to each location a predicate describing the flow condition, the invariant, and the urgency condition respectively. Here, $\mathcal{V}$ denotes the set of all variables, $\mathcal{V} = \{\{x \mid x \in V\} \cup \mathcal{V}\}$ denotes the set of variables plus their dotted versions, and $P(X)$ denotes the set of all predicates over the variables from set $X$.

$$E \in V \times P(\mathcal{V}) \times L_{\text{basic}} \cup CE(\mathcal{V}) \times (\mathcal{V}^d \times P(\mathcal{V} \cup \mathcal{V}^+)) \times V$$

denotes the discrete transitions (edges) of the automaton, where $L_{\text{basic}}$ denotes the set of basic action labels, $CE(\mathcal{V})$ denotes the set of all send and receive labels, formally defined as $CE(\mathcal{V}) = \{\text{h|es, h?xx} \mid h \in \mathcal{H} \land e_s \in \text{Expr}(\mathcal{V})^+ \land x \in V^d\}$ where $X^d$ denotes the set of lists whose elements are taken from set $X$, $\text{Expr}(X)$ denotes the set of all expressions over variables from set $X$, and $\mathcal{H}$ is the set of all channels, $2^X$ denotes the powerset of $X$, and $X^+ = \{v^+ \mid v \in X\}$ is used to refer to the values of the variables after the execution of an action.

The set of interchange automata $\mathcal{A}$ is defined by the following grammar:

$$\alpha ::= \alpha_{\text{atom}}$$

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\alpha$</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\parallel \alpha$</td>
<td>Parallel composition operator</td>
<td>$\gamma_0(\alpha)$</td>
</tr>
<tr>
<td>$\alpha \triangleright \alpha$</td>
<td>Synchronizing action operator</td>
<td>Initialization operator</td>
</tr>
<tr>
<td>$\llbracket y_0 = e_0, \dot{\bar{x}} = e_1 :: \alpha \rrbracket$</td>
<td>Variable scope operator</td>
<td>$\llbracket A \alpha :: \alpha \rrbracket$</td>
</tr>
<tr>
<td>$\llbracket H h :: \alpha \rrbracket$</td>
<td>Action scope operator</td>
<td>$\llbracket H h :: \alpha \rrbracket$</td>
</tr>
<tr>
<td>$\llbracket s_t(\alpha) \rrbracket$</td>
<td>Channel scope operator</td>
<td>$\llbracket s_t(\alpha) \rrbracket$</td>
</tr>
<tr>
<td>$\llbracket s_{\text{own}}(\alpha) \rrbracket$</td>
<td>Channel encapsulation operator</td>
<td>State variable operator</td>
</tr>
<tr>
<td>$\llbracket U_w(\alpha) \rrbracket$</td>
<td>Ownership operator</td>
<td>$\llbracket U_w(\alpha) \rrbracket$</td>
</tr>
<tr>
<td>$\llbracket D_{x;c}(\alpha) \rrbracket$</td>
<td>Urgent action operator</td>
<td>$\llbracket D_{x;c}(\alpha) \rrbracket$</td>
</tr>
</tbody>
</table>

where $\alpha \in L_{\text{basic}}$ is a basic action label; $u \in P(\mathcal{V})$ is a predicate over variables and dotted variables; $x$ is a variable, and $e \in \{\bot\} \cup \text{Expr}(\mathcal{V})$ is an expression, where $\bot$ denotes the undefined value; $h \in \mathcal{H}$ is a channel; $w \in L_{\text{basic}} \cup \mathcal{H}$ is a basic action label or a channel; and $G$ is a set of solutions (see Section 3.2 for more detail on solutions).

### 2.2 Informal semantics

The semantics of an interchange automaton is defined in terms of action transitions and time transitions between states consisting of the interchange automaton itself, a valuation of the variables and dotted variables of the automaton, and a set of state variables. These variables are the only ones that are not allowed to change arbitrarily in action transitions. By default all variables can change after the execution of an action.

The edges of the atomic interchange automaton are used to specify which actions can be executed. An edge $(v, g, l, (W, r), v')$ can be chosen to perform an action in a given valuation $\sigma$ if $v$ is the active location; the predicates $g, \text{flow}(v)$, and $\text{inv}(v)$ are satisfied in $\sigma$; and it is possible to find a new valuation $\sigma'$ in which the predicates $r, \text{flow}(v')$, and $\text{inv}(v')$ are satisfied. After the execution of the action specified by this edge, the active location is updated to $v'$.

Time can pass in an active location $v$ as long as the predicates $\text{flow}(v)$, $\text{inv}(v)$, and $\text{tcp}(v)$ hold, and no urgent actions become enabled. Section 3.2 describes how model variables are allowed to change when time passes. The active location is not modified.

*Parallel composition operator $\alpha_0 \parallel \alpha_1$ allows to execute two automata in parallel, which can interact by means of synchronizing actions, communication of values via channels, and shared variables. By default the actions of $\alpha_0$ and $\alpha_1$ are interleaved.*

*Synchronizing action operator $\gamma_0(\alpha)$ specifies that actions $a$ have to be synchronized with other actions named $a$ that were declared as synchronizing in a parallel context.*

*Initialization operator $\alpha \triangleright \alpha$ restricts the initial valuations in which $\alpha$ can start its execution to those which satisfy $u$.*

Variables, actions, and channels can be introduced using *scope operators*, which make the identifiers being declared invisible outside of the scope. In particular, the term $[\llbracket y_0 = e_0, \dot{\bar{x}} = e_1 :: \alpha \rrbracket]$ declares a local variable $x$ (and associated ‘derivative’ $\dot{x}$),
the initial value of which is given by the expression $e_0$ ($e_1$), unless $e_0 = \bot$ ($e_1 = \bot$) in which case $x$ ($\hat{x}$) can have any value.

Channel encapsulation operator $\partial_h(\alpha)$ forces send and receive actions via channel $h$ to execute synchronously, by blocking their individual occurrences.

State variable operator $st_1(\alpha)$ declares variable $x$ as a state variable for automaton $\alpha$, which means that $x$ cannot change arbitrarily in action transitions. In other words, if $\alpha$ does not change $x$ explicitly, for instance by means of an assignment, its value remains the same after the execution of an action.

Ownership operator $\text{own}_2(\alpha)$, declares the variable $x$ as belonging to automaton $\alpha$. This means that only $\alpha$ can change the value of $x$. Other automata cannot change the value of $x$, unless this change is allowed by $\alpha$ in a synchronizing action.

Urgent action operator $U_w(\alpha)$ declares action $w$ (or the send/receive actions via $w$ iff $w$ is a channel) as urgent. This means that time passing is allowed iff action $w$ is not enabled during the time interval.

2.3 Description of Hybrid Transition System

The structured operational semantics (SOS) of the CIF associates a labeled transition system (LTS) with every interchange automaton. There are three kind of transitions: action transitions, time transitions, and environment transitions.

An action transition $(\alpha, \sigma, X) \xrightarrow{\sigma, A} (\alpha', \sigma', X')$ models the execution of an action with label $l$, starting in state $(\alpha, \sigma, X)$, and resulting in a new state $(\alpha', \sigma', X')$. The set $A$ contains the set of synchronizing actions of $\alpha$, and the sets $X$ and $X'$ contain the set of state variables (see section 3.2) of $\alpha$ and $\alpha'$, respectively.

A time transition $(\alpha, \sigma, X) \xrightarrow{\rho, \theta, \phi} (\alpha', \sigma', X')$ models the passing of $t$ time units, starting in state $(\alpha, \sigma, X)$, and resulting in a new state $(\alpha', \sigma', X')$. Function $\rho$ contains for each variable the trajectories on time interval $[0, t]$, and function $\theta$ contains for each action the trajectory of the guard(s) associated with this action on time interval $[0, t]$.

A environment transition $(\alpha, \sigma, X) \xrightarrow{\sigma} (\alpha', \sigma', X')$ states that 1) $\sigma$ ($\sigma'$) is consistent with the initial conditions and active invariants of $\alpha$ ($\alpha'$); 2) the values of the variables owned by $\alpha$ remain unchanged; and 3) the set of synchronizing actions of $\alpha$ is $A$.

3. NEW CONCEPTS

This section presents the new concepts of the abstract CIF, along with the most important deduction rules. We have chosen to structure this section based on the concepts, introducing the most important deduction rules only. Furthermore, examples are specified using some abuse of notation. For example, an assignment to a variable $x$ on an edge of an automaton can be specified as $x := 1$, thus omitting the locations, guards and action labels.

3.1 Introducing variables, actions, and channels

In the concrete format, variables are introduced by means of closed scopes and open scopes. Both concepts of scoping can be found in modeling languages. In the abstract format, as defined in Beek et al. (2007b), however, the notion of an interchange automaton was formalized using concepts from hybrid automaton theory (e.g. see Alur et al. (1995)). This means that variables, action labels and channels were defined in interchange automata, and hiding operators were used instead of scoping operators.

This mismatch in the way variables were introduced in the abstract and concrete level added considerable complexity to the function that mapped the concrete CIF format to the abstract format. In the new CIF, the sets of internal and external variables have been removed from the atomic automata. External variable are introduced in the valuations, and internal variables are introduced using scope operators.

3.2 Action and delay behavior of variables

In the concrete format of the CIF, variables are declared as discrete, continuous or algebraic. Such a declaration defines the behavior of variables in the following way.

For actions, the value of a discrete or continuous variable does not change in an action transition unless it is explicitly specified to be allowed to change. E.g. incrementing the value of variable $x$ by one is expressed on an edge of an automaton by $\{ x : x := x + 1 \}$. The set of variables that is allowed to change is $\{ x \}$, and in predicates $x^+ := x + 1$, $x$ and $x^+$ refer to the values of $x$ before and after the transition, respectively. All other discrete and continuous variables are not allowed to change, and all other algebraic variables are allowed to change.

For delays, the value of a discrete variable is a constant function; continuous variables change according to a continuous function such that the solution function for $x$ is the integral of the solution function for $\hat{x}$ (and $\hat{x}$ is the derivative of $x$, if such a derivative exists); algebraic variables may behave according to a discontinuous function.

To allow an improved separation of concerns, the semantics of the discrete, continuous and algebraic variables is defined by two operators in the abstract CIF: the state variable operator defines the action behavior, and the dynamic type operator defines the delay behavior. Furthermore, the ownership operator $\text{own}(\alpha)$ provides new functionality: it prevents automata from making changes to variables that are owned by other automata.

State variable operator Application of the state variable operator $st_1(\alpha)$ adds $x$ to the set of state variables $X$, thus preventing changes in action transitions. This is formalized in rule 1:

$$ (\alpha, \sigma, X \cup \{ x \}) \xrightarrow{\sigma, A} (\alpha', \sigma', X') \xrightarrow{1} (st_1(\alpha), \sigma, X) $$

In principle, the translation function from the concrete CIF format to the abstract format ensures that the state variable operator is applied for the discrete and continuous variables.

Ownership operator Formally, this operator only affects environment transitions. Here, the variable that is declared as owned by the automaton is not allowed to change. This is used in parallel composition, see Section 3.5.

$$ X, X' \models (\alpha, \sigma) \xrightarrow{A} (\alpha', \sigma'), \sigma(x) = \sigma'(x) \xrightarrow{2} (\text{own}_2(\alpha), \sigma) \xrightarrow{\text{own}_2(\alpha'), \sigma'} $$
The difference between the ownership operator and the state variable operator is subtle; the state variable operator prevents a variable from changing during an action defined within the automaton itself, whereas the ownership operator prevents a variable from jumping during an action defined within another automaton.

*Dynamic type operator* The dynamic type operator $D_{x:G}(\alpha)$ has no effect on action and environment transitions. For time transitions, the operator ensures that the trajectories for the variable $x$ and the dotted version of the variable $\dot{x}$ is restricted to the behavior that is specified by the set of pairs of solution functions $G$. E.g. for a discrete variable $x$, the set $G$ would consist of pairs of a constant function as first element, and a zero function as second element. For a continuous variable, for each pair, the second function would be the derivative of the first (if existing), and the first would be the integral of the second.

$$X, X' \models (\alpha, \sigma) \xrightarrow{\gamma} (\alpha', \sigma'), (\rho \downarrow x, \rho \downarrow \dot{x}) \in G$$

3.3 Initialization

In the previously published abstract CIF, initialization was specified by means of the init predicate of an atomic interchange automaton. The init predicate was also used to store the values of local variables. The initialization predicate that was present in the ‘old’ interchange automata is specified by means of the initialization operator in the new abstract CIF, and the value of each local variable is stored in a corresponding variable scope operator. To capture the fact that all initialization predicates are taken into account simultaneously, as is the case in hybrid automata, the following properties should hold: $u \gg (u' \gg \alpha) \iff u \land u' \gg \alpha$ and $(u \gg \alpha) \parallel (u' \gg \alpha') \iff u \land u' \gg (\alpha \parallel \alpha')$. To ensure that the initialization predicate is removed after it has been taken into account, the deduction rules are:

$$X, X' \models (\alpha, \sigma) \xrightarrow{A} (\alpha', \sigma'), \sigma \models u$$

4

and likewise for time transitions. Rule 8 for the parallel composition ensures that when automaton $a_0$ executes an action, the initialization predicates of automaton $a_1$, executing in parallel, are also taken into account. Furthermore, the resulting (right hand side of the transition) parallel composition of the automata in the conclusion is constructed as the parallel composition of the resulting automata in the premises, to ensure that the initialization predicates are removed after they have been taken into account. Note that the environment transitions avoid the need for functions that depend on the syntax of the automata, which may endanger compositionality of the semantics.

3.4 Synchronization

In the previously defined CIF semantics, actions with the same name were synchronizing by default. However, common practice has shown that this is inconvenient, see Theunissen et al. (2008, 2009). This is illustrated by means of the following example.

Consider motor $M_0$ with two controllers $C_0$ and $C_1$. Controller $C_0$ stops the motor if it detects a low oil pressure. Similarly, controller $C_1$ stops the motor if it detects overheating. The motor and controllers can be modeled separately by the automata specified in Fig. 2. Using the previously defined semantics, the motor can be stopped only if an abnormal event is detected by both controllers, since the three automata synchronize on the ‘stop’ event.

In the revised CIF semantics, actions having the same name are non-synchronizing by default. If synchronization is intended, this can be achieved by using the synchronization operator $\gamma()$. Informally, the automaton $\gamma_\alpha(\alpha)$ declares the action $a$ as synchronizing, which means the the execution of action $a$ in $\alpha$ synchronizes with other automata that have also declared $a$ as synchronizing. Therefore, in $\gamma_\alpha(\alpha_0) \parallel \gamma_\alpha(\alpha_1)$ action $a$ must be executed simultaneously in both automata. If, on the other hand, in $\gamma_\alpha(\alpha_0) \parallel \alpha_1$, action $a$ is not declared as synchronizing in $\alpha_1$, then execution of actions $a$ in both automata will be interleaved. The desired behavior in the example represented in Fig. 2 can be specified in the new CIF by $\gamma_{\text{stop}}(\text{Motor}) \parallel \gamma_{\text{stop}}(\text{Controller}_0) \parallel \gamma_{\text{stop}}(\text{Controller}_1)$.

Note that synchronization by default, as defined in the previous CIF semantics, can be modeled in the revised abstract language by enclosing every atomic automaton by appropriate $\gamma()$ operators. The converse is, however, not true. Therefore, the revised abstract format is more expressive than the previously defined one.

The semantics of the synchronization operator is defined using the set of action labels from the action and environment transitions. Thus, the effect of the operator $\gamma_\alpha(\alpha)$ is to add the action $a$ to this set, as shown in the following rule:

$$X, X' \models (\alpha, \sigma) \xrightarrow{A} (\alpha', \sigma'), \sigma \models u$$

6

In the SOS rules for parallel composition, the set of synchronizing actions (on the action and environment transitions) is used to determine whether an action is synchronizing.

3.5 Parallel composition

Consider a parallel composition of two automata, each with two locations connected by means of one edge, with a shared synchronizing action $a$, and two shared variables $x$ and $y$. The automata assign the value 1 to the variables $x$ and $y$, respectively. With some abuse of notation this can be specified as $\gamma_\alpha(x : x^+ = 1) \parallel \gamma_\alpha(y : y^+ = 1)$. In the previously

<table>
<thead>
<tr>
<th>Motor:</th>
<th>start 0</th>
<th>stop 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller 0:</td>
<td>0 stop low pressure 1</td>
<td>stop 2</td>
</tr>
<tr>
<td>Controller 1:</td>
<td>0 stop overheating 1</td>
<td>step 2</td>
</tr>
</tbody>
</table>

Fig. 2. Motor and controller automata
The new syntax and semantics facilitates compositional verification, because the ownership operator can be applied to an automaton $\alpha$ in a parallel composition (e.g. $\text{own}_x(\alpha) \parallel \alpha'$) to prevent changes to its state variable $x$ by parallel automaton $\alpha'$.

### 3.6 Urgency

Urgent actions were introduced in timed and hybrid formalisms to allow easy modeling of greedy, or eager, behavior. In the CIF, the passing of time can be restricted by means of the tcp predicate, or by means of the urgent action operator $U_a(\alpha)$ that declares action $a$ to be urgent.

The following example shows a relation between the tcp predicate and the urgent action operator. Let $\alpha = ([v], v, [v \rightarrow true], [v \rightarrow true], [v \rightarrow true], [e])$ denote an atomic interchange automaton consisting of a single location $v$ with invariant, flow, and tcp predicates all true, and one self loop edge $e$ with guard time $\geq 1$ and action label $a: e = (v, time \geq 1, a, (\emptyset, true), v)$. Then defining the action $a$ as urgent by application of the urgent action operator $U_a(\alpha)$ is equivalent to the automaton $\alpha'$ obtained from $\alpha$ by replacing the tcp predicate by the negation of the guard: $\alpha' = ([v], v, [v \rightarrow true], [v \rightarrow true], [v \rightarrow ¬(time \geq 1)], [e])$. However, in the case that an action $a$ synchronizes in a parallel context, it is not possible to express urgency of such an action using tcp predicates in a compositional way.

The urgent action operator applied to a parallel composition of two automata $U_a(\alpha_0 \parallel \alpha_1)$ aims to express that time can pass only for as long as no action $a$ becomes enabled in $\alpha_0 \parallel \alpha_1$. For a synchronizing action $a$ to become enabled, the guards of the action $a$ need to be true in both automata $\alpha_0$ and $\alpha_1$. For a non-synchronizing action $a$ to become enabled, it is sufficient that a guard in one of the automata $\alpha_0$ or $\alpha_1$ is enabled. Note that time cannot pass anymore when an urgent action becomes enabled, independently of whether such an action can actually be executed. If an invariant of a target location would be false, then deadlock could be the result.

The required semantics is obtained by augmenting the time transitions by a pair of guard trajectories ($\theta_g, \theta_h$), as defined in the hybrid Chi formalism (see Beek et al. (2008a)). Namely $\theta_g$ for the synchronizing actions, and $\theta_h$ for the non-synchronizing actions. Note that for the purpose of simplicity, urgent channels are not discussed here. A guard trajectory is defined as a mapping from time-points to guard valuations: $\theta_g, \theta_h \in [0, t] \rightarrow ((L_{\text{basic}} \cup \{\top\}) \rightarrow B)$, and for all $s \in [0, t]$, $a \in \text{dom}(\theta_g(s))$, $x \in \{y, n\}$: $\theta_g(s)(a)$ iff the action $a$ is enabled (the value of the associated guard is true) at time $s$.

Guard trajectories are defined in the time transition rule of the atomic interchange automaton:

$$\forall s \in [0, t], \rho(s) = f(v_0) \land i(v_0), \forall s \in [0, t], \rho(s) = c(v_0), \rho(0) = \sigma, t > 0$$

$$((V, v_0, f, i, c, E), \sigma, X) \xrightarrow{\rho(t)} ((V, v_0, f, i, c, E), \rho(t), X)$$

where $\theta = (\theta_g, \theta_h)$ and for all time points $s \in [0, t]$ and actions $a \in L_{\text{basic}}$, $\theta_g$ and $\theta_h$ are defined as

$$\text{dom}(\theta_g(s)) = \emptyset$$

$$\theta_h(s)(a) = \rho(s)(\forall g \in g \in [i(v_0), f(W, x), \rho(t), E])$$

To be robust for variable abstraction, the value $\rho(s)(g)$ of guard $g$ is used in the guard valuation instead of the guard expressions itself. If one would choose to use the guard expressions,

1 Note that $\forall g \in g \in g$ denotes the predicate false
abstraction from variables may unexpectedly change the way in which actions are enabled and disabled, and hence may change the nature of the urgent behavior of those actions.

The urgent action operator $U_a(\alpha)$ allows for time passing as long as the guard valuation of the action $a$ remains false.

$$
(\alpha, \sigma, X) \xrightarrow{t, p, \sigma} (\alpha', \sigma', X').
$$

For synchronizing actions, the guard trajectory is defined as the conjunction of the guard trajectories $\theta_0$ and $\theta_1$, whereas for non-synchronizing actions, the resulting guard trajectory is defined as the disjunction of the guard trajectories $\theta_0$ and $\theta_1$.

Finally, for the synchronizing action operator $\gamma_0(\alpha)$ we can informally state that it ‘moves’ the guard trajectory for action $a$ from $\theta_0$ to $\theta_1$, resetting the guard trajectory for $a$ in $\theta_0$ to false.

4. CONCLUDING REMARKS

The main advantages of the redesign of the abstract format are as follows: the abstract and concrete languages are now based on similar concepts so that the transformation of the concrete language to the abstract language will be straightforward; the different concepts in the abstract language are easier to identify and understand, because each concept is defined by an operator in an orthogonal way; compositional verification and they thank the anonymous reviewers for helpful comments on the draft of this article.

ACKNOWLEDGEMENTS

The authors thank Pieter Cuijpers, Jasen Markovski, and Michel Reniers for their contributions in developing the abstract CIF, and they thank the anonymous reviewers for helpful comments on the draft of this article.

REFERENCES


