# A Formal Foundation for Distributed Workflow Execution Based on State Charts

Dirk Wodtke and Gerhard Weikum

Department of Computer Science, University of the Saarland
P.O. Box 151150, D–66041 Saarbruecken, Germany
E–mail: {wodtke,weikum}@cs.uni–sb.de, WWW: http://www–dbs.cs.uni–sb.de/

**Abstract.** This paper provides a formal foundation for distributed workflow executions. The state chart formalism is adapted to the needs of a workflow model in order to establish a basis for both correctness reasoning and run–time support for complex and large–scale workflow applications. To allow for the distributed execution of a workflow across different workflow servers, which is required for scalability and organizational decentralization, a method for the partitioning of workflow specifications is developed. It is proven that the partitioning preserves the original state chart's behavior.

## 1 Introduction

Workflow management is a rapidly growing research and development area of very high practical relevance [GHS95, Mo96, VB96, WfMC95, Sh96]. Typical examples of (semi–automated) workflows are the processing of a credit request in a bank, the editorial handling and refereeing process for papers in an electronic journal, or the medical treatment of patients in a hospital. Informally, a workflow consists of a set of activities with explicitly specified control and data flow between the activities. An activity may invoke a transaction or some specific application (e.g., a portfolio assessment program), or it may simply prompt a human user for intellectual work. The results of an activity are propagated back to the workflow and may drive the further control flow. Activities are assigned to execution roles, which are dynamically filled by automated or human actors depending on the required skills and competence. Presently, virtually all commercial systems and most of the research prototypes lack a mathematically rigorous foundation, so that formal reasoning about the correctness of workflow executions is infeasible.

### 1.1 Problem Statement

The specification of a workflow is usually done via high–level graphical interfaces, e.g., by drawing nodes and arcs. This specification must be mapped into an internal representation that serves as the basis for execution. In many workflow management systems, the underlying internal representation uses an ad hoc model and thus lacks capabilities for formal correctness reasoning. In contrast, general–purpose specification formalisms for dynamic systems such as Petri nets, state charts, temporal logic, or process algebras, which are pursued in various research projects and a few products, come with a rich theory and thus provide an excellent basis for formal proofs. For the work in this paper we have adopted the method of state and activity charts by Harel et al. [Ha87a, Ha87b, Ha88, Ha90, i–Log91, HN95], which is perceived by practitioners as more intuitive and easier to learn than Petri nets yet has an equally rigorous semantics. In particu-

lar, state chart specifications are amenable to model checking [McM93, HK94], so that critical workflow properties that are expressible in temporal logic can be formally verified; an example would be that a credit request must be rejected if it turns out that the customer has insufficient collaterals.

Formal reasoning about specifications implicitly assumes a centralized execution model; there is no notion of distribution, interoperating workflow engines, and so on. So tools for formal reasoning must have access to the complete workflow specification in a uniform representation, regardless of whether the workflow may in reality span different autonomous business units of an enterprise or even different enterprises. In fact, however, distributed, decentralized execution of workflows is a mandatory requirement for complex, large–scale applications for two reasons. First, such applications may involve a very large number of concurrent workflow instances which impose a high load on the workflow engine. Consequently, scalability and availability considerations dictate that the overall workflow processing be distributed across multiple workflow engines that run on different servers, and this workload partitioning may itself require the partitioning of individual workflows. Secondly, whenever a workflow spans multiple business units that operate in a largely autonomous manner, it may be required that those parts of a workflow that are under the responsibility of a certain unit are managed on a server of that unit. Thus, the partitioning and distribution of a workflow may fall out naturally from the organizational decentralization.

### 1.2 Contribution of the Paper

So there seems to be an inherent incompatibility between the centralized view of tools for formal reasoning and the need for decentralized, distributed execution in a real–life setting, and the question arises as to how significant or insignificant formal proofs are when their underlying model deviates substantially from the actual execution. This paper aims to reconcile the centralized specification and distributed execution views by providing theoretical underpinnings for transforming a specification into a distributed execution scheme within the formal framework of state charts. To this end, we proceed in three major steps and make the following contributions:

1. We first develop a simplified operational semantics of state and activity charts that is tailored to the use for workflow specification. We assume that a workflow can be specified completely in this ”canonical” form; specifications that are given in another language (e.g., in a scripting language such as FlowMark’s FDL) can be converted into our state and activity chart dialect.
2. We assume that activities are intellectually assigned to business units according to their roles and responsibilities. Then a state and activity chart is partitioned such that all activities and corresponding states of one business unit constitute one partition. In this transformation process, we make use of the notion of orthogonal components of a state chart, which is the standard form to express parallel execution: the resulting partitions form orthogonal components. This representation is directly amenable to distributed execution.
3. The transformation process is formally defined as a mapping between state charts, which we refer to as ”orthogonalization”. The paper’s major theoretical result is a theorem that this mapping is a homomorphism between state charts, which means that the transformation preserves the behavior of the specification. Thus, all formally derived properties of the centralized specification do provably carry over to the distributed execution scheme as well.

The work reported here is embedded in the MENTOR project [WWWK96a, WWWK96b] on "middleware for enterprise–wide workflow management" with emphasis on banking applications. This project uses the commercial state chart tool STATEMATE [Ha90, i–Log91] along with our own "glueing" software as a local workflow engine, and combines multiple instances of this engine for distributed execution using the TP monitor Tuxedo. The idea of transforming a centralized state chart specification into an equivalent collection of orthogonal components that is amenable to distributed execution has been introduced informally in [WWWK96a]; in the current paper we develop the formal underpinnings for this approach, thus justifying that centralized correctness reasoning and decentralized execution can indeed be reconciled. A formal approach with similar goals, albeit in a Petri net setting and unrelated to workflow, is presented in [Ho91]. However, this approach is extremely complex and restricted to simple place/transition nets (as opposed to richer Petri net variants such as predicate/transition nets).

The outline of the paper is as follows. Section 2 is an informal overview of state and activity charts. Section 3 gives the syntax and operational semantics of our simplified, workflow–specific state chart version. Section 4 presents the orthogonalization and partitioning of state charts, and develops the proof that this mapping is a homomorphism. Section 5 discusses further practical aspects of this approach. Due to length restrictions, some of the simpler definitions are given in an informal style in the paper. Full details can be found in [WW96].

## 2 Overview of the State Chart Specification Method

State and activity charts were originally developed for reactive systems (e.g., embedded control systems in automobiles) and have been quite successful in this area. They comprise two dual views of a specification.

*Activities* reflect the functional decomposition of a system and denote the "active" components of a specification; they correspond directly to the activities of a workflow. An *activity chart* specifies the data flow between activities, in the form of a directed graph with data items as arc annotations.

*State charts* reflect the behavior of a system in that they specify the control flow between activities. A state chart is essentially a finite state machine with a distinguished initial state and transitions driven by Event–Condition–Action rules (ECA rules). Each transition arc between states is annotated with an ECA triple. A transition from state X to state Y fires if the specified event E occurs and the specified condition C holds. The effect is that state X is left, state Y is entered, and the specified action A is executed. Conditions and actions are expressed in terms of data item variables, for example, those that are specified for the data flow in the corresponding activity chart; conditions and events may also refer to states and the modification of variables by means of special predicates like "in(s)" and "ch(v)" where s is a state and v is a variable. In addition, an action A can explicitly start or stop an activity and can generate an event E or set a condition C. ECA rules of this kind are notated in the form E[C]/A. Each of the three components may be empty. If all components are empty, then a transition can fire immediately; that is, a state change is possible without any precondition. Every state change in a state chart execution is viewed as one timestep; thus, state changes induce a discrete time dimension.

Two important additional features of state charts are *nested states* and *orthogonal components*. Nesting of states means that a state can itself contain an entire state chart. The semantics is that upon entering the higher–level state, the initial state of the embedded

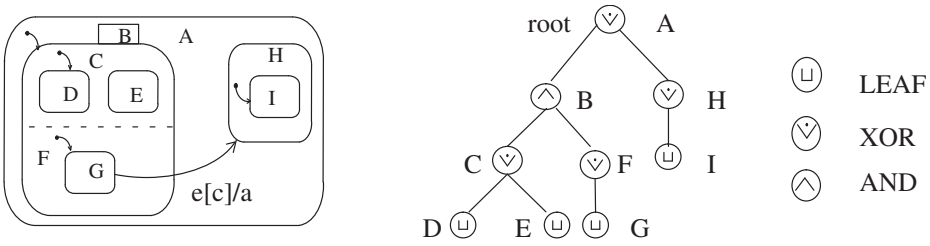


Fig. 1: Example of a state chart and its corresponding state tree

lower–level state chart is automatically entered, and upon leaving the higher–level state all embedded lower–level states are left. The possibility of nesting states is especially useful for the refinement of specifications during the design process, but this is not further considered in this paper. Orthogonal components denote the parallel execution of two state charts that are embedded in the same higher–level state (where the entire state chart can be viewed as a single top–level state). Both components enter their initial state simultaneously, and the transitions in the two components proceed in parallel, subject to the preconditions for a transition to fire.

The left portion of Fig. 1 shows an example chart with states A through I. Initial states are marked by an arc without source, and orthogonal components are separated by a dashed line within their corresponding higher–level state. For example, C and F are orthogonal components inside B, with D and G as their initial states. Only one of the transitions, namely, from G to H, has an explicit transition condition "e[c]/a".

## 3 Syntax and Operational Semantics of State Charts

In this section, we define the syntax and the operational semantics of a simplified version of state charts that is "tailored" to our application area of workflow management. Operational semantics of the full–fledged state chart formalism have been developed, for example, in [Ha87a, HN95, Hu88] (see [vB94] for a survey). However, these are unnecessarily complex for our purpose, mutually incompatible, and fall short of intuitive comprehensibility. For example, to cover a broad spectrum of application areas, an intricate timing model has been incorporated into these operational semantics that allows executing multiple "micro–step" actions within one time step. However, according to our experience with workflow applications, workflow specifications do not need the semantics of "micro steps" or related concepts (such as "chain reactions" within a step), but are served best by a "synchronous time model" in the sense of [HN95]. Thus, the following considerations, rather than inventing yet another state chart interpretation, aim to adapt the state chart formalism to the needs of a workflow model.

### 3.1 Syntax of State Charts

***Definition 1:*** A state tree S_tree is a triple (S, $type_S$, $act_S$), where
- S is a set of states,
- $type_S$ is a function which defines for each state its type:
  $type_S$: S → {AND, XOR, LEAF},
- $act_S$ is a function which maps each state onto its corresponding activity:
  $act_S$: S → Act.

The function $act_S$, which is not shown in Fig. 1 , defines an assignment of states to activities. In the workflow setting considered here, states capture the processing states of a

workflow (e.g., waiting for the third review in an electronic submission workflow). A state is assigned to the activity that is in progress or about to be activated when the state is entered; note that assigning a state to exactly one activity is not a restriction, since activities can also be nested according to their functional relationships.

***Definition 2:***

- A *transition label*, is composed of an event component, E, a condition component, C, and an action component, A, and is written as: E[C]/A. E and A may be empty; in this case or if C=true, the trivial components are omitted.
- A *transition* $t = (s_1, s_2, lbl) \in (2^S \times S \times L)$ is composed of:
  – a set of source states, $s_1$, which is either a singleton or empty (i.e., $|s_1| \leq 1$),
  – a target state, $s_2$,
  – a transition label, lbl.
- A transition is called a *default transition* if source(t)=∅.

The formal definition of the syntax of events, conditions, actions, and expressions is given in [WW96]. For example, there are events en(s) and conditions in(s) indicating that state s has been entered or is currently entered, respectively, and an event ch(el) indicating that element el has changed its value, and so on. These elements can be composed by using the operators of the propositional calculus to form expressions in a straightforward manner.

***Definition 3:*** A *state chart* is a pair (S_tree, T), where

– S_tree is a tree of states,
– T is a set of transitions such that for each non–leaf state s the following holds:
  • if s is of type XOR, there exists exactly one default transition which has one of its children as its target,
  • if s is of type AND, its children must not be targets of default transitions.

The children of AND type states are referred to as orthogonal components (e.g., C and F in Fig. 1).

Note that the definition of activity charts is straightforward and will not be needed explicitly in the following. Informally, activity charts differ from state charts in that they have data flow connectors between activities instead of transitions between states, and activities are assigned to “modules”, which in turn correspond to the servers of the various business units in an enterprise–wide workflow.

## 3.2 Operational Semantics of State Charts

We define the operational semantics of state charts in terms of “executing” a finite state machine, i.e., by identifying: (a) a set of states, (b) a state transition relation, (c) a set of initial states, and (d) a set of final states. With regard to state charts, parameter (a) is the set of *system configurations* which will be defined in Definition 10; parameter (b) is the *step function* which will be given by Definition 11; parameter (c) is the *initial system configuration* which will be defined in Definition 10; parameter (d) is not explicitly defined for state charts since the state chart execution does not necessarily need to terminate. Informally, we consider a state configuration as final when no transitions can fire anymore (regardless of further “external stimuli”, i.e, user input).

***Definition 4:*** Let S be a state chart. A *state configuration* cf of S is a set of states of S such that the following constraints hold:

– the root of the state tree of S is in cf,
– for each state which is in cf and of type AND, all its children are in cf,
– for each state which is in cf and of type XOR, exactly one of its children is in cf.

At each discrete point of time during the state chart execution, the state configuration denotes the set of currently entered states. The *initial state configuation*, $cf_0$, contains the root state, all children of type AND states which are in the initial state configuration, and one child of each type XOR state which is in the initial state configuration and which has children. In Fig. 1 the initial state configuration, $cf_0$, is {A, B, C, D, F, G}.

***Definition 5:*** Let S be a state chart. Given the sets of primitive (i.e., explicitly named) conditions, $C_p$, primitive events, $E_p$, and variables, $V_p$, a *context* ct of S is a mapping of these conditions, events, and variables onto values:

$ct : C_p \cup E_p \cup V_p \rightarrow \{true, false, generated\} \cup INT \cup \{ \perp \}$

where $\perp$ denotes the value "undefined".

The *initial context*, $ct_0$, is defined as false for primitive conditions and undefined (i.e., not generated) for primitive events and variables unless their initial values have not been set explicitly. For a given context, the values of expressions over variables and event as well as condition names can be computed inductively in a straightforward manner. This evaluation of expressions is done by the function *eval*.

A state configuration changes whenever at least one transition fires. In order to find out whether a transition fires at a given state configuration, two aspects need to be considered. First, one has to check whether the transition's source state is in the current state configuration. Second, it has to be determined whether the current values of the event and condition components of its transition label permit its firing, i.e., the event component has to evaluate to "generated" and the condition component has to evaluate to "true" under the eval function for the current context.

***Definition 6:*** Given a state configuration $cf_i$, and a context $ct_i$, $i \geq 0$, a transition t=(src, trgt, E[C]/A), $t \in T$, *fires at timestep i* if and only if one of the following conditions holds:

- if $src \in cf_i \wedge eval(E, cf_i, ct_i) = generated \wedge eval(C, cf_i, ct_i) = true$ (6.1)
- if src = ∅ and there exists a transition firing at the same step which has the type XOR parent of trgt as its target, (6.2)
- if src = ∅ and there exists a transition firing at the same step which has the type AND grandparent of trgt as its target. (6.3)

Informally, a transition fires in three cases: its source state is element of the current state configuration and its transition label evaluates to "generated" and to "true", respectively (6.1) (e.g., in Figure 1 the transition from G to H fires if G is in the current state configuration and the event and condition component of the transition label evaluate to "generated" and to "true", respectively); it is a default transition, and its target has a type XOR parent which is entered at the same timestep by the firing of another transition (6.2) (e.g., in Figure 1, the default transition which has I as its target fires when the transition from G to H fires); it is a default transition and its target has a type AND grandparent state which is entered at the same timestep by the firing of another transition (6.3). In the following, the set of transitions that fire at a given timestep i is denoted by $\tau_i$.

Resulting from the firing of transitions, a new state configuration is computed:

***Definition 7:*** Given a state configuration $cf_{i-1}$, a context $ct_{i-1}$, and the corresponding set of firing transitions $\tau_{i-1}$, $i > 0$, the *state configuration at timestep i*, $cf_i$, is inductively defined as follows:

- if $s \in cf_{i-1} \wedge \neg \exists_{t \in \tau_{i-1}} s \in source^*(t)$ then s is in $cf_i$, (7.1)
- if $\exists_{t \in \tau_{i-1}} s \in target^*(t)$ then $s \in cf_i$ (7.2)

where source*(t) and target*(t) are the sets of left and entered states, respectively, of the firing transitions. These two sets are formally defined in [WW96]. Informally, a state is an element of the new state configuration if it is already in the preceding state configuration and has not been left in timestep i–1 (7.1) or it is entered by the firing of a transition in timestep i–1 (7.2).

When a transition fires, the action component of its transition label is executed which may incur changes of the context by assigning new values to events, conditions, and variables. In addition to these assignments, users may generate external stimuli, i.e., generate events, set conditions, or write variables.

***Definition 8:*** Given a context $ct_{i-1}$ and a set of firing transitions $\tau_{i-1}$, i>0.

- The *external stimuli of timestep i* are defined as follows:
  $es_i : C_p \cup E_p \cup V_p \rightarrow \{true, false, generated\} \cup INT \cup \{ \perp \}$

$$es_i(x) = \begin{cases} generated, & \text{if } x \in E_p \wedge gen(x) \text{ has been entered at timestep i} \\ true & \text{, if } x \in C_p \wedge tr!(x) \text{ has been entered at timestep i} \\ false & \text{, if } x \in C_p \wedge fs!(x) \text{ has been entered at timestep i} \\ n & \text{, if } x \in V_p \wedge x{:=}n \text{ has been entered at timestep i} \\ \perp & \text{, otherwise} \end{cases}$$

- Let A(t) be the action component of transition t. The *assignments of timestep i* are defined as follows:
  $assign_i : C_p \cup E_p \cup V_p \rightarrow \{true, false, generated\} \cup INT \cup \{ \perp \}$

$$assign_i(x) = \begin{cases} generated, & \text{if } x \in E_p \wedge \exists_{t \in \tau_{i-1}} gen(x) \in A(t) \wedge es_i(x) = \perp \\ true & \text{, if } x \in E_p \wedge \exists_{t \in \tau_{i-1}} tr!(x) \in A(t) \wedge es_i(x) = \perp \\ false & \text{, if } x \in E_p \wedge \exists_{t \in \tau_{i-1}} fs!(x) \in A(t) \wedge es_i(x) = \perp \\ n & \text{, if } x \in V_p \wedge \exists_{t \in \tau_{i-1}} (x := n) \in A(t) \wedge es_i(x) = \perp \\ es_i(x) & \text{, if } es_i(x) \neq \perp \\ \perp & \text{, otherwise} \end{cases}$$

In summary, assignments can be initiated by user interactions, in the case that es(x) is defined, or by the execution of actions that are part of transition labels of firing transitions. Note that the es(x) assignment has higher priority than the assignment initiated by the firing of a transition. Resulting from these assignments a new context of a state chart has to be computed.

***Definition 9:*** Given the sets of conditions, $C_p$, events, $E_p$, and variables, $V_p$, the *context at timestep i* is defined as follows:

$ct_i : C_p \cup E_p \cup V_p \rightarrow \{true, false, generated\} \cup INT \cup \{ \perp \}$

$$ct_i(x) = \begin{cases} assign_i(x), & \text{if } assign_i(x) \neq \perp & (9.1) \\ \perp & \text{, if } assign_i(x) = \perp \wedge x \in E_p & (9.2) \\ ct_{i-1}(x) & \text{, otherwise} & (9.3) \end{cases}$$

The context changes if there are explicit assignments due to user interaction or actions in transition labels (9.1). In addition, all events that have not been generated are set to undefined (i.e., “not generated”) (9.2). This means that the generation of events is only

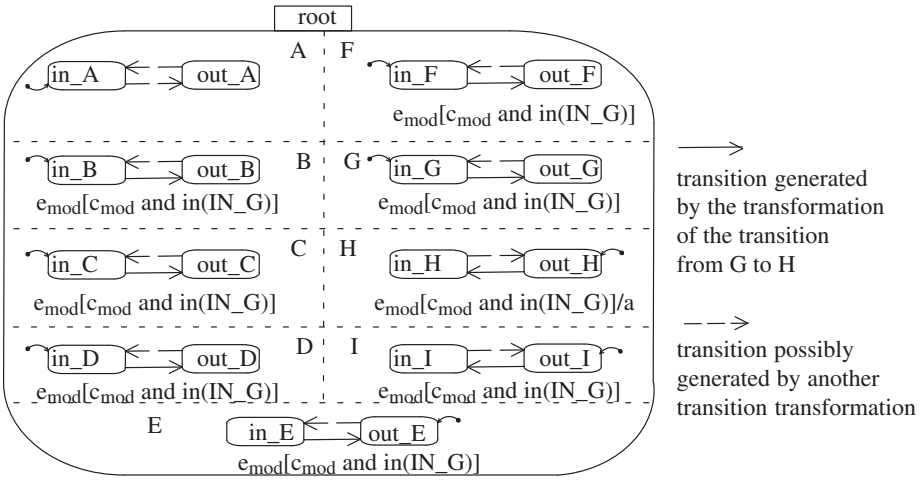


Fig. 2: Transformation of the state chart of Fig. 1

valid for one step. In contrast to events, the context of conditions and variables that are not subject to assignments is the same as in the previous step (9.3).

Now we are ready to define concisely the stepwise execution of a state chart.

***Definition 10:*** Given a state configuration $cf_i$ and a context $ct_i$, $i \geq 0$. The *system configuration* is defined as $sc_i := (cf_i, ct_i)$. The set of system configurations is denoted by $SC := \{sc_i | i \geq 0\} \subseteq CF \times CT$ where CF is the set of all possible state configurations and CT is the set of all possible contexts. The *initial system configuration* is denoted as $sc_0$.

***Definition 11:*** Let SC be a set of system configurations. The *step* function is defined as follows:

$step : SC \rightarrow SC$

$step : sc_i \mapsto sc_{i+1} \quad , i \geq 0$

***Definition 12:*** A *state chart execution* is a sequence of system configurations: $sc_0$, $sc_1$, $sc_2$, ... $sc_n$, with $step(sc_i) = sc_{i+1}$, $i \geq 0$.

# 4 Orthogonalization and Partitioning of State Charts

In this section we present our method for the partitioning of workflow specifications. The partitioning transforms a state chart into a behaviorally equivalent state chart that is directly amenable to distributed execution in that each distributable portion of the state chart forms an orthogonal component. The first step towards a partitioned specification is thus called “orthogonalization”. The outcome of this step is another state chart that can easily be partitioned. The proof that this transformation is indeed feasible in that it preserves the original state chart’s behavior is cast into showing that the resulting “orthogonalized” state chart is a homomorphic image of the original specification. Note that we do not address the issue of “streamlining” or optimizing business processes by altering the invocation order of activities. We are solely concerned with preparing workflow specifications for distributed execution.

## 4.1 Orthogonalization of State Charts

The basic idea of the orthogonalization is that each state s of the original chart is simulated by an orthogonal component with two states in_s and out_s, thus keeping track of entering and leaving s. While this is straightforward, the technical difficulty is to

adapt the transitions in the correct way. This involves augmenting the labels of the original chart's transitions by additional predicates of the form "in(in_s)". Consider the example of Figure 1. The corresponding orthogonalized chart is shown in Figure 2. For each of the original states, A through I, orthogonal components with identical names are generated, each one consisting of a pair of states, two interconnecting transitions, and a default transition. Next, the orthogonalization which is done in three steps, i.e., the state transformation, the generation of default transitions, and the transformation of the transitions, will be described in detail.

***Definition 13:*** Given a state tree, S_tree, with $S=\{s_1, ..., s_n\}$ being its set of nodes. The *orthogonalized state tree* S_tree' := $ortho_{state}$(S_tree) is a state tree of height 3 with $S':=\{root, s_1, ..., s_n, in_s_1, ..., in_s_n, out_s_1, ..., out_s_n\}$ being its set of nodes and the following types, parent/child relationships, and assignments to activities:
type(root)=AND, type(s)=XOR, type(in_s)=type(out_s)=LEAF,
parent(s)=root, parent(in_s)=parent(out_s)=s,
$act_{S'}(in_s)=act_{S'}(out_s)=act_{S'}(s) := act_S(s)$.

For each state s a triple of states, {s, in_s, out_s}, is generated where s becomes both child of the AND–type root of the orthogonalized state chart and XOR–type parent of in_s and out_s. The three generated nodes keep the assignment, $act_S$, of the original node.

***Definition 14:*** Given the orthogonalized state tree with states $S':=\{root, s_1, ..., s_n, in_s_1, ..., in_s_n, out_s_1, ..., out_s_n\}$ and the initial state configuration, $cf_0$, of a state chart, the set of *generated default transitions*, $T_{default}$' is the following set of transitions on S': $T_{default}' = \{ t' \mid source(t') = \emptyset \wedge target(t') = in_s \wedge s \in cf_0\} \cup \{ t' \mid source(t') = \emptyset \wedge target(t') = out_s \wedge s \notin cf_0\}$

Informally, an "in"–state becomes the target of a default transition if its corresponding state of the original state chart is element of the initial state configuration. Otherwise the corresponding "out"–state becomes the target of a default transition. Since in the example of Figure 1 $cf_0$ is {A, B, C, D, F, G}, in Figure 2 the target states of default transitions are in_A, in_B, in_C, in_D, in_F, in_G, and out_E, out_H, out_I.

***Definition 15:*** Given a state chart with transition set T. Let S' be the state set of the corresponding orthogonalized state tree and let for each transition $t=(s_j, s_k, E[C]/A)\in T$ $source^*(t)$ and $target^*(t)$ be the sets of left and entered states when t fires. The *orthogonalized transition set* is $ortho_{transition}$(T) with: $ortho_{transition}(T):=$

$$\bigcup_{t\in T}\left\{\begin{array}{ll}\{(in_s, out_s, E_{mod}[C_{mod}'] \mid s \in source^*(t)\} \cup & (15.1)\\ \{(out_s_k, in_s_k, E_{mod}[C_{mod}'] / A)\} \cup & (15.2)\\ \{(out_s, in_s, E_{mod}[C_{mod}']) \mid s \in target^*(t),\ s \neq s_k\} & (15.3)\end{array}\right\}$$

with: $C_{mod}'$:= $C_{mod}$ and $in(in_s_j)$ (15.4)
$C_{mod}$:= C where all terms of the form "in(s)" are replaced by "in(in_s)" (15.5)
$E_{mod}$ := E where all terms of the form "en(s)" or "ex(s)" are replaced by "en(in_s)" or "ex(in_s)", respectively (15.6)

Informally, each transition t is mapped onto at least two transitions. For each state which is left when t fires a transition which has the corresponding "out"–state as its target is generated (15.1) (e.g., in Figure 2 these are the transitions which interconnect the pairs of states (in_B, out_B), (in_C, out_C), (in_D, out_D), (in_E, out_E), (in_F, out_F), (in_G, out_G)). In addition, one transition is generated which has the corresponding "in"–state of target(t) as its target (e.g., in Figure 2 this is the transition interconnecting

out_H and in_H). Note that this is the only transition which inherits the action component, a, from t (15.2). In addition, for each other state which is entered when t fires a transition which has the corresponding "in"–state as its target is generated (15.3) (e.g., in Figure 2 this is the transition interconnecting out_I and in_I).

Concerning the transition labels, the transition transformation extends the condition component of each transition label by the conjunctive term "in($s_j$)" with $s_j$ denoting the source state of t (15.4). Terms in event and condition components which are related to states are replaced by terms with adjusted state names (15.5), (15.6) (e.g., "en(state)" is changed to "en(in_state)"). In the example of Figure 2, the event and condition components of the transition label "e[c]/a" are extended to "$e_{mod}[c_{mod}$ and in(IN_G)]".

The above considerations are summarized in the following definition of an orthogonalized state chart.

***Definition 16:*** Let s_chart = (S_tree, T) be a state chart, and let $ortho_{state}$(S_tree), $T_{default}$', $ortho_{transition}$(T) be the orthogonalized state tree, the generated set of default transitions, and the orthogonalized transition set of s_chart as defined above.
The *orthogonalized state chart* s_chart':=ortho(s_chart) is defined as (S_tree',T') with S_tree' := $ortho_{state}$(S_tree) and T' := $ortho_{transition}(T) \cup T_{default}$'.

## 4.2 The Homomorphism Property

Now we are about to state the main result of the paper, namely, that the orthogonalized state chart of a given state chart preserves the original chart's behavior. There are several notions of "behavioral equivalence" discussed in the literature on reactive and concurrent systems (see, e.g., [MP92, Mil89]). However, we can exploit the specific nature of our simplified state chart version to introduce a simpler and, in our opinion, more elegant notion of behavioral equivalence. Intuitively, behavioral equivalence of two state charts means that both produce the same externally observable actions under the same external stimuli, where the observable actions in a workflow setting would be the invocation of workflow activities along with their actual parameters. This is certainly ensured if the two state charts exhibit equivalent transition sequences under the same external stimuli. The transition sequences in turn are uniquely determined by the step function of Definition 11, i.e., the operational semantics that we have defined and which holds uniformly for all state charts including those that result from the orthogonalization transformation. Now, by viewing a state chart as an algebraic structure with its system configurations as its carrier set and the step function as a unary operator, we can ensure equivalent transition sequences of two state charts by defining a homomorphism between the two algebras. Thus, we have to define a mapping between the system configurations of two state charts such that this mapping and the step operator commute.

***Definition 17:*** Consider a state chart with state set, S={$s_1$, ... , $s_n$}, the set of state configurations, CF, the set of transitions, T, and the set of contexts, CT. Let S', CF', T', CT' be the corresponding sets of the orthogonalized state chart. The mappings $h_{CF}$, $h_{CT}$, and $h_{SC}$ are defined as follows:

– *mapping of state configurations:* $h_{CF}$: CF→CF' (17.1)
$h_{CF}(cf)=cf'$ with cf':={root, $s_1$, ... , $s_n$}∪{in_s|s∈ cf}∪{out_s|s∉ cf},

– *mapping of contexts:* $h_{CT}$: CT→CT' (17.2)
$h_{CT}(ct)=ct$ i.e., the identity mapping,

– *mapping of system configurations:* $h_{SC}$: SC→SC' (17.3)
$h_{SC}(sc)=sc'$ with: sc = (cf, ct), sc' := ($h_{CF}$(cf), $h_{CT}$(ct)).

For example, the initial state configuration of the state chart in Figure 1, {A, B, C, D, F, G}, is mapped onto the state configuration of the state chart of Figure 2, {root, A, B, C, D, E, F, G, H, I, in_A, in_B, in_C, in_D, in_F, in_G, out_E, out_H, out_I}.

The result that we are heading for is that the mapping $h_{SC}$ is a homomorphism. We therefore introduce a step operator of the orthogonalized chart which we denote by step'. However, note that both the step operator of the original state chart and step' are defined by the generic function of Definition 11. It has to be proved that the following holds: $h_{SC}(step(sc)) = step'(h_{SC}(sc))$.

In the following, we proceed through a series of lemmas towards our main theorem. The proofs of the lemmas 1 and 2 and the full proof of Lemma 3 are contained in [WW96].

***Lemma 1:*** Given a state chart, s_chart. Let s_chart' be a a state chart which differs from s_chart only in that the condition component of each transition label is further constrained in the following way: Each transition t=(source, target, E[C]/A) is changed into t'=(source, target, E[C and in(source)]/A). Let $t \in \tau_i$, $t' \in \tau_i'$ be firing transitions in timestep i of s_chart and s_chart', respectively. The following holds for the sets of firing transitions, $\tau$ and $\tau'$, of s_chart and s_chart', respectively: $t \in \tau_i \Leftrightarrow t' \in \tau_i'$ for all $i \geq 0$.

The lemma guarantees that by constraining the condition components of the transitions in the described manner, the transitions fire exactly at the same timestep as the corresponding original transitions. However, the question that needs to be answered is: Do the constrained transitions have the same effects on the state chart execution as the corresponding original transitions?

***Lemma 2:*** Consider a state chart, s_chart. Let s_chart' be a state chart which differs from s_chart only in that the condition component of each transition label is further constrained as described in Lemma 1. Let $t \in \tau_i$, $t' \in \tau_i'$ be firing transitions in timestep i of s_chart and s_chart', respectively. Let EG(t), CST(t), CSF(t), AE(t) be the sets of generated events, the sets of conditions set to true and set to false, and the set of actions executed as a result of the firing of t, respectively. Let EG(t'), CST(t'), CSF(t'), and AE(t') be the according sets for transition t'. The following holds: EG(t) = EG(t'), CST(t) = CST(t'), CSF(t) = CSF(t'), AE(t)=AE(t') for all $t \in \tau_i$, $t' \in \tau_i'$ for all $i \geq 0$.

According to the lemma, the firing of transitions with constrained transition labels actually has the same effects as the corresponding original transitions; in particular, the same actions are executed (e.g., the same activities are started at the same timestep). From these lemmas we obtain:

***Corollary 1:*** Let SC be the set of system configurations of a state chart s_chart. Let SC' be the set of system configurations of s_chart' which differs from s_chart only in that the condition component of each transition label is further constrained by adding the conjunctive term "in(source)" where source is the source state of the transition. The following property holds: For all timesteps $i \geq 0$, under the condition that $es_i(x)=es_i'(x)$ it holds: $sc_i=sc_i'$.

Informally, the meaning of the corollary is that by constraining the condition components of the transition labels the behavior of a state chart does not change. Next, we will concentrate on the relationship between the operational semantics (i.e., the step function) of a state chart and its orthogonalized representation.

***Lemma 3:*** Let $cf_i$ be the state configuration of a state chart s_chart at timestep $i \geq 0$. The state configuration of the orthogonalized state chart s_chart' at the same timestep is $h_{CF}(cf_i)$.

***Proof Sketch (Lemma 3)*** We use induction to prove the lemma.

- Basis of the induction: The initial state configuration of s_chart', $cf_0$', is $h_{CF}(cf_0)$ where $cf_0$ is the initial state configuration of s_chart.
  Proof: Let the set of states of the original state chart be $S=\{s_1, ..., s_n\}$. Because of Definitions 13 and 14, the initial state configuration of the orthogonalized state chart, $cf_0$', is: $\{root, s_1, .., s_n\} \cup \bigcup_{s \in cf_0}\{in_s\} \cup \bigcup_{s \notin cf_0}\{out_s\}$.
  This is exactly the set of states according to Definition 17.1, namely $h_{CF}(cf_0)$.
- Induction hypothesis: For each timestep i>0 the state configuration of s_chart', $cf_i$', is $h_{CF}(cf_i)$ with $cf_i$ being the state configuration of s_chart at timestep i.
- The induction step is as follows: $cf_{i+1}' = h_{CF}(cf_{i+1})$
  Generally, $cf_{i+1}$ is the state configuration that is reached from $cf_i$ when the transitions in $\tau_i$ fire. Therefore,

$$h_{CF}(cf_{i+1}) = h_{CF}\left(cf_i \cup \bigcup_{t \in \tau_i}\{target^*(t)\} - \bigcup_{t \in \tau_i}\{source^*(t)\}\right)$$

  For the computation of the right side of the above equation, we will use the following evidently correct computation rule for $h_{CF}$:
  Let $S=\{s_1, ..., s_n\}$ be a set of states, and let $S_1:=\{s_1, ..., s_k\}$, $S_2 := S_1 \cup \{s_{k+1}\}$, and $S_3 := S_1 - \{s_k\}$ be subsets of S. The following computation rules hold for $h_{CF}$:
  – $h_{CF}(S_2) = h_{CF}(S_1 \cup \{s_{k+1}\}) = h_{CF}(S_1) - \{out_s_{k+1}\} \cup \{in_s_{k+1}\}$
  – $h_{CF}(S_3) = h_{CF}(S_1 - \{s_k\}) = h_{CF}(S_1) - \{in_s_k\} \cup \{out_s_k\}$
  Applied to the above equation we obatin the following equation:

$$h_{CF}(cf_{i+1}) = h_{CF}(cf_i) - \bigcup_{t \in \tau_i}\{out_s | s \in target^*(t)\} \cup \bigcup_{t \in \tau_i}\{in_s | s \in target^*(t)\} - \bigcup_{t \in \tau_i}\{in_s | s \in source^*(t)\} \cup \bigcup_{t \in \tau_i}\{out_s | s \in source^*(t)\}$$

  Because of the induction hypothesis, this is equivalent to:

$$h_{CF}(cf_{i+1}) = cf_i' - \bigcup_{t \in \tau_i}\{out_s | s \in target^*(t)\} \cup \bigcup_{t \in \tau_i}\{in_s | s \in target^*(t)\} - \bigcup_{t \in \tau_i}\{in_s | s \in source^*(t)\} \cup \bigcup_{t \in \tau_i}\{out_s | s \in source^*(t)\}$$

  According to Definition 15, we substitute the above four unions. The outcome is as follows shown below:

$$\begin{aligned} h_{CF}(cf_{i+1}) = cf'_i &- \bigcup_{t' \in ortho_{transition}(t), t \in \tau_i}\{out_s | target(t') = out_s\} \\ &- \bigcup_{t' \in ortho_{transition}(t), t \in \tau_i}\{in_s | source(t') = in_s\} \\ &\cup \bigcup_{t' \in ortho_{transition}(t), t \in \tau_i}\{in_s | target(t') = in_s\} \\ &\cup \bigcup_{t' \in ortho_{transition}(t), t \in \tau_i}\{out_s | source(t') = out_s\} \end{aligned} \quad (*)$$

  According to Definition 7, $cf_{i+1}$' can be computed as follows:

$$cf_{i+1}' = cf_i' - \bigcup_{t' \in \tau_i'}\{source^*(t')\} \cup \bigcup_{t' \in \tau_i'}\{target^*(t')\}$$

  Since, generally, $source^*(t) = source(t) \wedge target^*(t) = target(t)$ holds for all orthogonalized state charts and for all transitions t, this is equivalent to:

$$cf_{i+1}' = cf_i' - \bigcup_{t' \in \tau_i'}\{source(t')\} \cup \bigcup_{t' \in \tau_i'}\{target(t')\}$$

  Both "out"–states and "in"–states can be elements of source(t') and target(t'). Therefore, we can write:

$$\begin{aligned} cf_{i+1}' = cf_i' &- \bigcup_{t' \in \tau_i'}\{out_s | source(t') = out_s \in cf_i'\} \\ &- \bigcup_{t' \in \tau_i'}\{in_s | source(t') = in_s \in cf_i'\} \\ &\cup \bigcup_{t' \in \tau_i'}\{in_s | target(t') = in_s \notin cf_i'\} \end{aligned}$$

$$\cup \bigcup_{t' \in \tau_i'} \{out_s | target(t') = out_s \notin cf_i'\} \quad (**)$$

Two cases have to be considered:

case 1: $\tau_i' = ortho_{transition}(\tau_i)$

Obviously, in this case (*) and (**) are identical, i.e., $h_{CF}(cf_{i+1}) = cf_{i+1}'$ holds.

case 2: $\tau_i' \neq ortho_{transition}(\tau_i)$

Two cases have to be considered: First, there exists a transition in the orthogonalized state chart which fires at timestep i, however the corresponding transition in the original state chart does not fire. Secondly, there exists a transition which fires in the original state chart at timestep i whereas the corresponding transitions in the orthogonalized state chart, or at least one of them, does not fire.

case 2.1: Because of Definition 15, for each transition of the original state chart at least two transitions have been generated by the orthogonalization. Without loss of generality, we assume that by applying $ortho_{transition}$ to a transition t=(s, r, E[C]/A) four transitions, t'={$t_1$', $t_2$', $t_3$', $t_4$'}, have been generated. All transitions in t' have the same event and condition components, $E_{mod}$ and $C_{mod}$', in their transition labels. In addition, all condition components contain the conjunctive term "in(in_s)". Suppose that for one transition, e.g., $t_1$', the following holds: $t_1' \in \tau_i'$, $t_1' \notin ortho_{transition}(\tau_i)$. From $t_1' \in \tau_i'$ it follows: in_s$\in cf_i'$, eval($E_{mod}$, $cf_i'$, $ct_i'$)=generated and eval($C_{mod}$', $cf_i'$, $ct_i'$)=true $\Rightarrow$ eval(in(in_s), $cf_i'$, $ct_i'$)=true . Because of the assumption of case 2.1, $t \notin \tau_i$ must hold; however, from $ct_i'=ct_i$ and from the induction hypothesis it follows eval(E, $h_{CF}(cf_i)$, $ct_i'$)=generated, eval(C, $h_{CF}(cf_i)$, $ct_i'$)=true. $\Rightarrow$ To prevent t from firing s$\notin cf_i$ must hold. Because of the induction hypothesis, $cf_i'=h_{CF}(cf_i)$, it follows: in_s$\in cf_i'$. This is a contradiction.

case 2.2: Without loss of generality, we suppose that by applying $ortho_{transition}$ to a transition t=(s, r, E[C]/A) four transitions, t'={$t_1$', $t_2$', $t_3$', $t_4$'}, have been generated. We assume that the following holds: t'={$t_1$', $t_2$', $t_3$', $t_4$'}$\subset \tau_i'$, t' $\subseteq$ {$ortho_{transition}(\tau_i)$}, with $t_1$'=(in_s, out_s, $E_{mod}[C_{mod}']$), $t_2$'=(out_r, in_r, $E_{mod}[C_{mod}']$/A), $t_3$'=(in_q, out_q, $E_{mod}[C_{mod}']$), $t_4$'=(out_p, in_p, $E_{mod}[C_{mod}']$). We assume that q is an element of s_r(t) and p is an element of t_r(t). In order to meet the assumption, it has to be proved that at least one of the transitions in t' cannot fire.

From t$\in \tau_i$ it follows: s$\in cf_i$, eval(E, $cf_i$, ct)=generated, eval(C, $cf_i$, $ct_i$)=true. From the induction hypothesis we know that in_s$\in cf_i'$ holds. Furthermore, if there are any expressions in event or condition components of the transition labels of transitions in t' which refer to states, like en(s), in(s), etc., these expressions are translated into according "in"–expressions, e.g., en(in_s), in(in_s), etc. By this argument and knowing that in_s$\in cf_i'$, and $ct_i=ct_i'$, it follows: eval($E_{mod}$, $cf_i'$, $ct_i'$)=generated and eval($C_{mod}$', $cf_i'$, $ct_i'$)=true

Considering Corollary 1, this means that the only thing that can prevent one of the transitions in t' from firing is that its source state is not in $cf_i'$. Since in T' are only transitions which interconnect "in"–states and "out"–states, this would imply that at least one of the following states must be in $cf_i'$: out_s or in_r or out_q or in_p. However, (1) with regard to $t_1$', because of the induction hypothesis in_s is in $cf_i'$, (2) with regard to $t_2$', if in_r would be in $cf_i'$ then r would be in $cf_i$ because of the induction hypothesis; but if s and r would be both elements of $cf_i$ their least common ancestor would have to be a state of type AND; this would disallow that there existed a transition interconnecting s and r $\Rightarrow$ out_r$\in cf_i'$, (3)

with regard to $t_3$', if out_q would be in $cf_i$' then q could not be an element of $s_r(t) \Rightarrow in_q \in cf_i$', (4) with regard to $t_4$', if in_p would be in $cf_i$' then p could not be an element of $t_r(t) \Rightarrow out_p \in cf_i$'. $\Rightarrow t' \subseteq \tau_i$ '. This is a contradiction.

Since both case 2.1 and case 2.2 lead to contradictions, the proof is completed.

Next, we will state a relevant property of $h_{SC}$ that will be exploited in the proof of the main theorem.

***Lemma 4:*** $h_{SC}$ is injective.

***Proof (Lemma 4)***

We will first prove that $h_{CF}$ is injective.

(Proof by contradiction) Assumption: $h_{CF}(cf_1)=h_{CF}(cf_2)$ and $cf_1 \neq cf_2$. Assuming that S is the set of states of the original state chart, it follows from the assumption: $\exists_{s \in S}\, s \notin cf_1 \wedge s \in cf_2$.

From $s \notin cf_1$ it follows: $out_s \in h_{CF}(cf_1) \wedge in_s \notin h_{CF}(cf_1)$; from $s \in cf_2$ it follows: $out_s \notin h_{CF}(cf_2) \wedge in_s \in h_{CF}(cf_1)$. This is in contradiction to the first assumption, $h_{CF}(cf_1)=h_{CF}(cf_2) \Rightarrow h_{CF}$ is injective.

Assuming that sc is (cf, ct), we know from Definition 17 that $h_{SC}(sc)$ is defined as the tuple $(h_{CF}(cf), h_{CT}(ct))$. Since $h_{CF}$ is injective and $h_{CT}$ is the identity mapping, it follows that $h_{SC}$ is injective.

We are now prepared for a corollary describing the relationship between the sets of firing transitions of the original and the orthogonalized state chart. This result follows directly from Lemma 3.

***Corollary 2:*** Let $\tau_i$ be the set of firing transitions of a state chart s_chart at timestep $i \geq 0$. The set of firing transitions of the orthogonalized state chart s_chart' at the same timestep is $\tau_i' := ortho_{transition}(\tau_i)$.

Corollary 1, Lemma 3, Lemma 4, and Corollary 2 lead us to the following main theorem.

***Theorem 1:*** The mapping $h_{SC}$ which maps the system configurations of a state chart onto the system configuration of the corresponding orthogonalized state chart is a homomorphism; that is, for all timesteps $i \geq 0$ and for all system configurations $sc_i$ of the original state chart the equation $h_{SC}(step(sc_i)) = step'(h_{SC}(sc_i))$ must hold, where step is the step function of the original state chart and step' that of the orthogonalized state chart (both defined by the operational semantics developed in Section 3).

***Proof (Theorem 1):*** Since the step operator is actually a composite function, the formal derivation of the homomorphism needs to refer to the state configuration, cf, and the context, ct. It has to be shown that, for all state configurations, $cf_i$, and for all contexts, $ct_i$, $step(cf_i, ct_i)$ results in the same system configuration as $h_{SC}^{-1}(step'(h_{SC}(cf_i, ct_i))$.

Consider a state chart, s_chart=(S_tree, T), its state set, $S = \{s_1, \ldots, s_n\}$, and its set of transitions, $T = \{t_1, \ldots, t_m\}$. Let $cf_i = \{s_1, \ldots, s_p\}$ be the current state configuration, and let $ct_i$ be the current context. First, we will orthogonalize the state chart: s_chart':=ortho(s_chart)=(S_tree', T') with $S_tree' := ortho_{state}(S_tree)$, $T' := ortho_{transition}(T) \cup T_{default}'$, and the transformed set of states, $S' := \{root, s_1, \ldots, s_p, in_s_1, \ldots, in_s_n, out_s_1, \ldots, out_s_n\}$. The proof is organized in four working steps:

<u>working step (1):</u> First, we will apply the step function to $cf_i$ and $ct_i$, i.e., compute $step(cf_i, ct_i)$:

Based on $cf_i$ and $ct_i$, the set of firing transitions, $\tau_i$, can be computed. Suppose that $\tau_i$ is $\{t_k, \ldots, t_l\}$. The firing of these transitions effects some states to be left: $\bigcup_{t \in \tau_i} \{source^*(t)\}$ and some states to be entered: $\bigcup_{t \in \tau_i} \{target^*(t)\}$. Without loss of generality, we assume that the set of left states is $\{s_1, \ldots, s_{j-1}\}$, and the set of entered

states is $\{s_{p+1}, ... , s_q\}$. Therefore, $cf_{i+1}$ is $\{s_j, ... , s_p, s_{p+1}, ... , s_q\}$. For the computation of the context $ct_{i+1}$, we assume that $CST(\tau_i)$, $CSF(\tau_i)$, $EG(\tau_i)$, and $EA(\tau_i)$, i.e, the sets of conditions set true, set false, the set of generated events, and the set of executed actions which result from the execution of the action components of the firing transitions, and the set of external stimuli $\{es_{i+1}(x)\}$ are known. Because of Definition 9, the new context, $ct_{i+1}$, is well–defined. Therefore, $step(cf_i, ct_i) = (cf_{i+1}, ct_{i+1})$.
working step (2): Now, we will apply $h_{CF}$ to $cf_i$ and $h_{CT}$ to $ct_i$:
$cf_i' := h_{CF}(cf_i) = \{s_1, ... , s_n, root, in_s_1, ... , in_s_p, out_s_{p+1}, ... out_s_n\}$, $ct_i' := h_{CT}(ct_i) = ct_i$.
working step (3): Now, we will apply the step function to $cf_i'$ and $ct_i'$:
First, we will compute the set of firing transitions of the orthogonalized state chart, $\tau_i'$. Because of Corollary 2, this set is $\tau_i'=ortho_{transition}(\tau_i)$.
With regard to the state configuration, we know from Definition 15 that each transition which is element of any $ortho_{transition}(t)$, $t \in \{t_k, ... , t_l\}$, has either a source state which is element of $\{in_s \mid s \in source^*(t) \cap cf_i\}$ or a target state which is element of $\{in_s \mid s \in target^*(t)\}$. Since all transitions in T' and thus all transitions in $\tau_i'$ interconnect pairs of "in"–states and "out"–states, the transitions in $\tau_i'$ interconnect the following pairs of states: $(in_s_1, out_s_1,)$ , ..., $(in_s_{j-1}, out_s_{j-1})$ and $(out_s_{p+1}, in_s_{p+1})$, ... , $(out_s_q, in_s_q)$. Since exactly all transitions in $\tau_i'$ fire, the states $in_s_1$, ..., $in_s_{j-1}$, $out_s_{p+1}$, ... , $out_s_q$ have to be deducted from the state configuration and the states $out_s_1$, ..., $out_s_{j-1}$, $in_s_{p+1}$, ... , $in_s_q$ have to be added to the state configuration. Therefore, the resulting state configuration is:
$cf_{i+1}'=\{s_1, ... , s_n, root, out_s_1, ... , out_s_{j-1}, in_s_j, ... , in_s_p, in_s_{p+1}, ... , in_s_q, out_s_{q+1}, ... , out_s_n\}$.
With regard to the context, the following holds: Updates of the context can only have their origin in the execution of action components of firing transitions or in external stimuli. The latter are the same as for the original state chart. Therefore, it remains to be proved that the same actions are executed: Since $ortho_{transition}(\tau_i)$ maps each firing transition of s_chart onto at least a pair of firing transitions in $\tau_i'$ and generates no additional action components of transition labels, and since for each action component of s_chart exactly one (identical) action component is generated by $ortho_{transition}$, the actions that are executed at timestep i in s_chart' are the same as in s_chart. Therefore, for the sets of conditions set true and false and for the set of generated events the following holds: $CST(\tau_i')=CST(\tau_i)$, $CSF(\tau_i')=CSF(\tau_i)$, $EG(\tau_i')=EG(\tau_i)$ and $EA(\tau_i')=EA(\tau_i)$.
working step (4): Now, we will exploit the injectivity of $h_{CF}$ (Lemma 4) in order to map $sc_{i+1}'$ onto $sc_{i+1}$: $cf_{i+1} = h_{CF}^{-1}(cf_{i+1}') = h_{CF}^{-1}(\{s_1, ... , s_n, root, out_s_1, ... , out_s_{j-1}, in_s_j, ... , in_s_p, in_s_{p+1}, ... , in_s_q, out_s_{q+1}, ... , out_s_n\}) = \{s_j, ... , s_p, s_{p+1}, ... , s_q,\}$
$ct_{i+1} = h_{CT}^{-1}(ct_{i+1}') = ct_{i+1}$
Since the outcome of working step (4) and working step (1) are identical, $h_{SC}$ is a homomorphism for all timesteps $i \geq 0$.

## 4.3 Partitioning of State and Activity Charts

The outcome of the orthogonalization is a state chart which consists of a set of orthogonal components. The final partitioning which transforms an orthogonalized state chart into a form that is amenable to distributed execution then falls out in a natural and straightforward manner. The orthogonalized state chart is decomposed along the boundaries of its orthogonal components, and all the resulting components with the same assignment to activities, $act_S$, are grouped together to form a new state chart.

In many cases, it is natural to further combine multiple activities into a coarser unit that corresponds to one business unit (e.g., a department). Assuming that each activity has been assigned to such a business unit all activities that belong to a business unit together with the corresponding state chart form a partition of the workflow.

Such partitions are the unit of distribution; that is, each partition may reside on a different server. Obviously, servers may have to communicate local state and context information to each other in order to ensure a synchronized evaluation of their local transition labels. A foolproof, still unoptimized solution would be that all servers mutually exchange all modified conditions, events, and variables after every time step. Using standard compiler techniques for data flow analysis, these communication steps can be restricted to the cases where the progress at one server may in fact potentially influence other servers. More details on this synchronization are discussed in [WWWK96a]. Note that in contrast to the orthogonalization, the correctness proof for the final partitioning is straightforward and thus omitted.

## 5 Practical Perspectives

Now that we have developed the formal underpinnings of our transformation approach, we will briefly discuss their applicability to a concrete workflow scenario. According to our experience in a collaboration with the Union Bank of Switzerland, state charts are still too formal in flavor and, therefore, not appropriate for wide use in a business environment. Rather high–level design tools for business process engineering or re–engineering (BPR) are preferred; these tools are geared for ease of use in the design process itself but cannot produce executable workflows. With this in mind, we suggest the following workflow design procedure: Workflow or business process specifications can be initially developed with BPR tools or the specification tools of the user's favorite commercial workflow system. Such a specification can be automatically converted into a state and activity chart specification (which is also shown in the figure). Next, the orthogonalization is carried out. Finally, the partitions that result from the orthogonalization are assigned to workflow servers, taking into account both the performance and availability requirements and the organizational responsibilities of the enterprise (e.g., one server for each involved department). Thus, the role of state and activity charts in a practical business environment, as we view it, is that of a canonical representation for the underlying execution engine (with rigorously defined semantics), into which other specifications can be converted and which may even serve as an exchange format across different workflow engines.

With regard to the correctness of state chart based workflow specifications, we have collected first practical experience with verification techniques. In particular, we have studied the validation of state chart properties by means of reachability tests (as supported by STATEMATE [Ha90]) and symbolic model checking (using the tool described in [HK94]). For example, a property that every credit request will eventually be granted or rejected, and that these two results exclude each other, can be easily expressed in the temporal logic CTL [Em90] and efficiently verified by means of model checking. On the other hand, given the inherent limitations of such verification techniques, we believe that a combination of verification and simulation is a viable approach towards higher confidence in the correctness of workflow specifications. The transformation approach developed in this paper contributes to this goal in that it justifies that verification and simulation can be applied to the centralized specification, and their results carry over to the actually distributed execution.

## References


[Em90] E.A. Emerson, Temporal and Modal Logic, in: J. van Leeuwen (ed.), Handbook of Theoretical Computer Science, Elsevier, 1990

[GHS95] D. Georgakopoulos, M. Hornick, A. Sheth, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, 3(2), 1995

[Ha87a] D. Harel, On the Formal Semantics of Statecharts, Proc. Symposium on Logics in Computer Science, Ithaca, New York, 1987

[Ha87b] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming Vol.8, 1987

[Ha88] D. Harel, On Visual Formalisms, Communications of the ACM, 31(5), 1988

[Ha90] D. Harel et al., STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering, 16(4), 1990

[HN95] D. Harel, A. Naamad, The STATEMATE Semantics of Statecharts, Technical Report, i–Logix Inc., October 1995

[HK94] J. Helbig, P. Kelb, An OBDD–Representation of Statecharts, Proc. European Design and Test Conference, 1994

[Ho91] R.P. Hopkins, Distributable nets, in: Rozenberg (ed.), Advances in Petrinets, LNCS 524, Springer, 1991

[Hu88] C. Huizing, Modelling Statecharts in a fully abstract way, in: Proc. CAAP, LNCS 299, Springer, 1988

[i–Log91] i–Logix Inc., Documentation for the Statemate System, 1991

[MP92] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer, 1992

[McM93] K.L. McMillan, Symbolic Model Checking, Kluwer, 1993

[Mil89] R. Milner, Communication and Concurrency, Prentice Hall, 1989

[Mo96] C. Mohan, Workflow Management Systems: State of the Art on Research and Products, Tutorial Notes, Int. Conf. on Extending Database Technology, Avignon, 1996

[Sh96] A. Sheth (ed.), Proc. NSF Workshop on Workflow and Process Automation in Information Systems, Athens, 1996, http://lsdis.cs.uga.edu/activities/NSF–workflow/

[vB94] M. von der Beeck, A Comparison of Statechart Variants, in: Formal Techniques in Real–Time and Fault–Tolerant Systems, LNCS 863, Springer, 1994

[VB96] G. Vossen, J. Becker (eds.), Business Process Modelling and Workflow Management – Models, Methods, Tools (in German), International Thomson, 1996

[WW96] D. Wodtke, G. Weikum, A Formal Foundation for Distributed Workflow Execution Based on State Charts, Technical Report, University of Saarbruecken, 1996

[WWWK96a] D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, The Mentor Project: Steps Towards Enterprise–wide Workflow Management, Proc. 11th International Conference on Data Engineering, New Orleans, 1996

[WWWK96b] J. Weissenfels, D. Wodtke, G. Weikum, A. Kotz Dittrich, The Mentor Architecture for Enterprise–wide Workflow Management, in: [Sh96]

[WfMC95] Workflow Management Coalition, 1995, http://www.aiai.ed.ac.uk/WfMC/