Abstract

Many formal modelling notations for business processes have been proposed during the last decade. They can be broadly classified into high-level visual notations, with an intuitive meaning, mainly addressed to the business management community and low-level foundational notations, with a detailed and formal semantics, mainly addressed to the computer science community. Role activity diagrams are a popular high-level visual notation for capturing the dynamics and role structure of an organization. This paper establishes that role activity diagrams have a formal semantics as well and thus making them suitable to formal verification. The result is obtained by mapping of a role activity diagram model to a process algebra model. Process algebras are mathematical languages for the specification and understanding of concurrent and cooperating computational processes.

1. Introduction

An increased interest in applying information technology to business process modelling has been manifested during the last decade in both business management and computer science communities. Because organizations are very complex artifacts, it has been claimed that carefully developed models are necessary for describing, analyzing and/or enacting the underlying business processes ([20]). As a result, many formal notations for modelling business processes have been proposed. They were classified in high-level visual notations, mainly addressed to the business management community and low-level foundational formalisms, mainly addressed to the computer science community ([4]). The second class was mostly inspired by the work on formal methods and the similarity that has been advocated between concurrent and cooperative computational processes and business processes. Note however that this similarity is a kind of conjecture that has never been proved, but instead has been quietly accepted and adopted.

Role activity diagrams (RAD hereafter) are a popular high-level visual language for capturing the dynamics and role structure of an organization ([20]). The RAD notation has been adopted in many applications involving business processes and has proved useful for various tasks like modelling ([19], [8]), simulation ([17]) and enactment ([21]). The reference work [20] states that RAD models have a formal semantics as well. However, to the best of our knowledge, there are no references in the literature reporting on the formal semantics of RAD. Moreover, in [20] it is given only some intuition regarding how a RAD model could be interpreted in terms of token playing using an analogy with Petri nets.

The aforementioned facts motivate the goal of this paper, i.e. to provide a formal semantics to RAD models. The work can be seen as an attempt to bridge the gap between high-level and low-level formalisms for business process modelling. The strategy we have adopted to achieve this desiderate was to look for an appropriate foundational formalism and then map RAD models to this formalism.

Process algebras are a family of mathematical languages for the specification of concurrent and cooperating computational processes. They have a respectable research community, a rich literature and various practical applications ([23], [16], [7], [6], [14], [22], [1]). Software tools to support modelling with process algebras are available ([16]). Our choice is also motivated by their following specific features: i) they allow the building of models using a compositional approach; ii) the resulting models are suitable for quantitative and qualitative analysis ([10], [24]).

The research reported in this paper continues the work started in the Inspire project on the formal analysis of business processes ([6], [3]).

The paper is structured as follows. Section 2 gives an overview of the RAD notation and introduces a modelling example. For an extensive introduction to the notation and
its practical application the reader is advised to consult the reference work [20]. Section 3 introduces the Finite State Process formalism (FSP hereafter) that we have chosen as target for our mapping. For a detailed exposition of FSP the reader is referred to the text [16]. Section 4 gives a concise definition of RAD models and an algorithmic description of the mapping of a RAD model to a FSP model. Section 5 suggests some uses of the resulting FSP model. Section 6 provides a discussion and some pointers to relevant related work. The last section concludes the paper.

2. Overview of RAD

RAD is a very popular visual notation specially tailored for business process modelling. It originates from the work on coordination in programming environments ([12]). The reference textbook that provides an extensive introduction to the notation is [20]. RAD were used in various applications reported in the literature ([8], [17], [21]). In what follows we shall briefly introduce the RAD concepts and graphical notation. The basic building blocks of the RAD notation are presented in figure 1.

Roles group together activities into units of responsibility, according to the set of responsibilities they are carrying out. A role has one or more execution threads containing sequential activities, parallel activities – part refinement and choices – case refinement. Roles are like classes in object orientation, i.e. they describe the behaviour of a class of role instances. A business process may contain one or more active instances of the same role. An actor is an agent that acts a role instance. An agent can be either a human or machine, single or group, which is capable of carrying out the work specified in the role.

Activities are the basic building blocks of a role. An activity can be carried out in isolation within a role or it may require coordination with activities in other roles. In the last case the activity is called interaction and it requires the involvement of all the participating roles (two or more) when it is carried out.

External events are points at which state changes occurring in the process environment influence on our process.

Roles have states. Carrying out the activities of a role can be seen as moving from state to state. States are useful to model point wise process goals, i.e. when a particular state was reached it means that a certain goal has been fulfilled.

Figure 2 shows a model of a business process for carrying out a design project. For a detailed description of the business perspective of this example see [20].

1Note that RAD supports also dynamic role instantiation, but this feature is not considered in this paper.
3. Overview of FSP

FSP is an algebraic specification technique of finite state labeled transition systems (LTS hereafter).

**Definition 1 (labeled transition system)** Let $\mathcal{S}$ be the universal set of states, $\mathcal{L}$ be the universal set of action labels and $\tau$ the internal unobservable action. A finite LTS is a quadruple $P = (\mathcal{S}, \mathcal{A}, \Delta, q)$ where:

i) $\mathcal{S} \subseteq \mathcal{S}$ is a finite set of states.

ii) $A = \alpha P \cup \{\tau\}$, where $\alpha P \subseteq \mathcal{L}$ denotes the alphabet of $P$.

iii) $\Delta \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ denotes a transition relation that maps from a state and an action onto another state.

iv) $q \in \mathcal{S}$ is the initial state of $P$.

LTS models are suitable for specifying various classes of discrete-event systems. However, the descriptions, either visual or textual, of LTS models as labeled directed graphs are impractical for more than a few states. For this reason the FSP process algebra has been proposed ([16]). FSP uses six operations: prefix, choice, parallel composition, hiding, re-labeling and definition.

i) Prefix. The process $a \rightarrow P$ performs the action $a$ and then behaves like $P$. The prefix operator specifies sequential execution of actions.

ii) Choice. The process $P \mid Q$ behaves either like $P$ or like $Q$. If both are enabled then the choice is non-deterministic.

iii) Parallel composition. The composite process $P \parallel Q$ specifies the interaction between processes $P$ and $Q$ on the common set of actions in their alphabets $\alpha P$ and $\alpha Q$. This means that for actions outside the set $\alpha P \cap \alpha Q$, $P$ and $Q$ proceed independently, but for actions in $\alpha P \cap \alpha Q$, $P$ and $Q$ must cooperate and thus proceed together.

iv) Hiding. The application of this operator to a process term has the effect of hiding some of the behaviour of the process from an external observer. The process $P \setminus H$ where $H$ is a set of actions, behaves like $P$, excepting that any action in set $H$ appears to an external observer as $\tau$. Hiding is an abstraction mechanism useful for restricting the behaviour of a process to the set of observable actions.

v) Re-labeling. Re-labeling functions applied to a process term change the names of the action labels. The process $P/L$ where $L = \{nl_1/o_1l_1, \ldots, nl_k/o_1l_k\}$, $o_1l_i \in L$, $nl_i \subseteq L$, behaves like $P$ excepting that any action $nl_i$ appears to an external observer as any of the actions in the set $nl_i$, for all $1 \leq i \leq k$.

vi) Definition. A definition $A = P_A$ associates the behaviour of the process term $P_A$ with the name $A$. You can then use $A$ in process terms to describe more complex behaviours. Thus $A$ is interpreted as the name of a re-usable process component.

The empty process that engages in no further actions is denoted by $END$.

Defining the syntax of FSP models is done in two steps. The key point is to not allow the arbitrary mixing of choices and parallel compositions in order to preserve the finiteness of the model state space ([16]). In the first step we define sequential processes and in the second step composite processes.

The set of process names is partitioned into the sets $\mathcal{P}_S$ of sequential process names and $\mathcal{P}_C$ of composite process names. Let $\mathcal{L}$ be the set of all action labels.

**Definition 2 (sequential process)**

i) A sequential process term is defined according to the following rules: a) $END$ is a sequential process term; b) If $SPN \in \mathcal{P}_S$ then $SPN$ is a sequential process term; c) If $a_i \in \mathcal{L}$ and $SP_i$ are sequential process terms, $1 \leq i \leq k$, then $\langle a_1 \rightarrow SP_1 | a_2 \rightarrow SP_2 | \ldots | a_k \rightarrow SP_k \rangle$ is a sequential process term.

ii) A sequential process definition of the sequential process with name $SPN_i$ is a sequence $SPN_1 = SP_1, \ldots, SPN_p = SP_p$ such that $SPN_i \in \mathcal{P}_S$ and $SP_i$ are sequential process terms. $1 \leq i \leq p$.

**Definition 3 (composite process)**

i) A composite process term is defined with the following rules: a) If $PN \in \mathcal{P}_C \cup \mathcal{P}_S$ then $PN$ is a composite process term; b) If $CP_i$ are composite process terms, $1 \leq i \leq k$, then $(CP_1 \parallel CP_2 \parallel \ldots \parallel CP_k)$ is a composite process term; c) If $CP$ is a composite process term and $L$ is a re-labeling function then $CP/L$ is a composite process term.

ii) A composite process definition that defines the composite process $CPN \in \mathcal{P}_C$ has the form $CPN = CP$ or $CPN = CP \setminus \{l_1, \ldots, l_p\}$, $l_i \in \mathcal{L}$, $1 \leq i \leq p$.

---

2FSP is more elaborated than what is shown here. For these details see [16]. However, the FSP subset introduced here is sufficient for the purpose of this paper.

3The elements of $\mathcal{P}_C$ are distinguished from the elements of $\mathcal{P}_S$ by prefixing with $\parallel$.

4The definitions of $SPN_2, \ldots, SPN_p$ are called local definitions.
Figure 3. The conditional interaction between a buyer and a seller

The second form of a composite process definition in point ii) of this definition corresponds to a hiding operation. A FSP model consists of a finite set of sequential and/or composite process definitions. Obviously, it is required that no process name occurring in the right-hand side of a composite process definition was left undefined.

Consider a business process displaying a conditional interaction between a buyer and a seller ([20], pp.83). The buyer orders some goods from the seller. After receiving the goods the buyer checks if they are ok and if they are not then it returns the goods back to the seller. After receiving the faulty goods, the caller will ship new goods to the buyer. The model is shown in figure 3.

In [20] it is described an informal semantics of RAD models in terms of animating tokens in a process model. Tokens sit on state lines and move from state to state. If we look at a state then we can see tokens entering it (action $P$) followed by the state being active (action $\delta$) and finally the tokens leaving the state (action $E$). Alternatively the state line may end. We are using the $\text{end} \rightarrow END$ term to specify a final state for all state lines. Thus the FSP model of a state line is:

\[
NL = L0, \\
L0 = (i \rightarrow L1|\text{end} \rightarrow END), \\
L1 = (s \rightarrow o \rightarrow L0).
\]

Sometimes we would like our model to be able to hide the fact that a specific state has been reached thus focusing only on input-output behaviour, i.e. token production and consumption. For this purpose we are using the hiding operator. The FSP model is:

\[
|L = NL \setminus \{s\}.
\]

States $B_0$, $S_0$, and $S_2$ in figure 3, for which there are no entering activities, are called start states. States $B_3$, $S_1$ and $S_4$ in figure 3, for which there are no leaving activities, are called end states. In terms of token moving, a start state acts as a token producer and an end state acts as a token consumer.

An end state consumes input tokens and at some point in time it ends.

\[
E = (i \rightarrow E|\text{end} \rightarrow END).
\]

As concerning start states, it is important to distinguish between start states that are entered a single time when the process is initiated – single instance start states and start states that are expected to be re-entered multiple times during the same execution of a process – multiple instance start states.

To illustrate this distinction, consider again the example shown in figure 3. When the process starts we assume there are a single buyer and a single seller and thus start states $B_0$ and $S_0$ are single instance. They are entered a single time, when the process is initiated. However, $S_2$ is expected to be re-entered whenever the buyer detects that the received goods are not ok and thus it is a multiple instance start state. In general, such situations are encountered in service interactions when a RAD fragment on the server side can be started/instantiated an indefinite number of times as requests for service come in ([20]).

A single instance start state generates a token and ends. Its FSP model is:

\[
SS = (o \rightarrow \text{end} \rightarrow END).
\]

A multiple instance start state repeatedly generates tokens and at some point in time it ends. Its FSP model is:

\[
SM = (o \rightarrow SM|\text{end} \rightarrow END).
\]

A role can be represented as a composition of state models. For the roles shown in figure 3 we obtain the composite processes \textbf{Buyer} and \textbf{Seller}. The system process is the composition of the processes of its component roles. The synchronization of the events for consuming/producing tokens ensures that the activities in the FSP representation of the RAD model occur in the correct order.
Figure 4. LTS of the process from figure 3 under the single-instance execution assumption

FSP has an operational semantics given via a LTS. The mapping of a FSP term to a LTS is described in detail in [16]. Figure 4 shows the LTS for the process shown in figure 3, under the single-instance execution assumption.

4. Mapping RAD to FSP

In order to define the mapping of a RAD model to FSP we need a rigorous definition of the concept of RAD model. A role of a RAD is a bipartite directed graph. The set of nodes of the graph is partitioned in two sets: i) the set of states that correspond to the states from the RAD notation and ii) the set of actions that correspond to the activities, external events, case refinements and part refinements from the RAD notation.

Some comments are necessary regarding the set of actions. For each activity and external event clearly we have one action node in the graph model. For a case refinement with \( n \) alternatives with conditions \( c_1, \ldots, c_n \) we have \( n \) action nodes, each one labelled with a condition \( c_i, 1 \leq i \leq n \). For each part refinement we have one action node. Moreover, if two or more threads originating from the same part refinement recombine together later then we add an action node needed to synchronize the incoming threads. An example is the situation in figure 2. Here we must synchronize the two threads in role Designer that originate from the part refinement that follows the execution of activity Agree TOR and delegate.

**Definition 4 (RAD model)**

i) Let \( A \) be a finite set of action nodes, \( S \) be a finite set of state nodes and \( E \subseteq (A \times S) \cup (S \times A) \) be a set of directed edges. A role model is a directed graph \( R = (A, S, E) \) such that no action nodes have an empty set of in coming or out coming edges.

ii) A RAD model is a finite set of role models \( R = \{ R_1, \ldots, R_k \} \).

For example, the role model corresponding to the role Buyer from figure 3 is shown in figure 5.

The set \( A \) of actions of a role is partitioned into the sets \( A_e, A_a, A_c, A_p, A_s \). \( A_e \) is the set of external events, \( A_a \) is the set of activities, \( A_c \) is the set of conditions from case refinements, \( A_p \) is the set of part refinements and \( A_s \) is the set of synchronization points due to re-combinations of threads originating from the same part refinement. Formally \( A_p = \{ a \in A \mid |E(a)| \geq 2 \}, A_s = \{ a \in A \mid |E^{-1}(a)| \geq 2 \} \) and \( A_c = \bigcup_{(s \in S) \wedge (E(s) \neq \emptyset)} E(s) \).

The set \( S \) of states is partitioned into the sets \( S_l \) and \( S_u \). \( S_l \) are the states labelled with state descriptions and \( S_u \) are the states corresponding to unlabelled state lines. The distinction between \( S_l \) and \( S_u \) is important in the mapping algorithm. Additionally, we have to distinguish in \( S \) the start states \( S_s \) and the end states \( S_e \). Formally \( S_s = \{ s \in S \mid E^{-1}(s) = \emptyset \} \) and \( S_e = \{ s \in S \mid E(s) = \emptyset \} \).

To each state \( s \in S \) we associate a process \( P(s) \) according to five cases:

i) if \( s \in S_u \) then \( P(s) = L/\{ E^{-1}(s)/i, E(s)/o \} \).

ii) if \( s \in S_s \) is a single instance start state then \( P(s) = SS/\{ E(s)/o \} \).

iii) if \( s \in S_s \) is a multiple instance start state then \( P(s) = SM/\{ E(s)/o \} \).

iv) if \( s \in S_e \) then \( P(s) = E/\{ E^{-1}(s)/i \} \).

---

\(^5\)The picture shown in figure 4 has been generated using the Labeled Transition System Analyzer tool ([16]).

\(^6\)If \( E \subseteq V \times V \) is a binary relation on \( V \) then for all \( v \in V \), \( E(v) = \{ w \mid (v, w) \in E \} \) and \( E^{-1}(v) = \{ w \mid (w, v) \in E \} \).
Figure 5. The graph corresponding to the role Buyer shown in figure 3

To each role model $R$ with the set of states $S(R)$ we associate a composite process $P(R) = (|s \in S(R) \rightarrow P(s))$. To each RAD model $\mathcal{R} = \{R_1, \ldots, R_k\}$ we associate a composite process $P = (|1 \leq i \leq k \rightarrow P(R_i))$. The resulting FSP model of $\mathcal{R}$ will contain the definitions of $L$, $NL$, $SS$, $SM$ and $E$ from section 3 and for all $s \in S(R_i)$ and $1 \leq i \leq k$, the definitions of processes $P(s)$ and $P(R_i)$ and the definition of the overall composite process $P$. RAD2FSP is an algorithm for computing the set $\Delta$ of FSP definitions from a RAD model $\mathcal{R}$.

**Algorithm RAD2FSP**

**Input**: a RAD model $\mathcal{R}$

**Output**: a FSP model $\Delta$ for $\mathcal{R}$

\[
\Delta \leftarrow \text{definitions of processes } L, NL, E, SS, SM
\]

$PN \leftarrow$ a new composite process name

$F_1 \leftarrow PN \cdot \gamma' = \gamma'$

For each $R \in \mathcal{R}$ do

$RN \leftarrow$ a new composite process name

$F_2 \leftarrow RN \cdot \gamma' = \gamma'$

For each $s \in S(R)$ do

$(SN, F_3) \leftarrow S2FSP(s)$

$\Delta \leftarrow \Delta \cup \{F_3\}$

$F_2 \leftarrow F_2 \cup \{SN\}$

If $s$ is not the last in $S(R)$ then $F_2 \leftarrow F_2 \cdot \gamma'$

$F_2 \leftarrow F_2 \cdot \gamma'$

$\Delta \leftarrow \Delta \cup \{F_2\}$

$F_1 \leftarrow F_1 + RN$

If $R$ is not the last in $\mathcal{R}$ then $F_1 \leftarrow F_1 \cdot \gamma'$

$F_1 \leftarrow F_1 \cdot \gamma'$

End for each $s \in S(R)$

End for each $R \in \mathcal{R}$

End algorithm RAD2FSP

Some clarifications of the RAD2FSP algorithm are necessary. The operator $+$ is string concatenation. The algorithm $S2FSP$ computes the FSP definition $F_3$ and the name $SN$ of the process $P(s)$ associated to state $s$ according to the five identified cases. In variable $F_1$ we compute the definition of the overall composite process. In variable $F_2$ we compute the definition of the composite process associated to each role.

**Proposition 1** The time complexity of the algorithm RAD2FSP for mapping a RAD model to FSP is $O(\sum_{R \in \mathcal{R}} |S(R)|)$.

**5. Possible uses of the resulting FSP model**

The resulting FSP model of a business process is useful for performing the following tasks: i) interactive simulation ([14]); ii) static verification of qualitative properties using “model checking” techniques ([13], [9], [14]); iii) dynamic
simulation for performance evaluation ([17]). These tasks are very important for business process analysis, design and re-engineering. Of course, in order to be able to carry out these tasks, appropriate software tools are necessary. A nice tool is LTSA and it is described in [16].

Interactive simulation allows you to trace into a process by executing it step by step. Even this feature is of limited use for serious applications, it still gives a glimpse of how the process might behave before having actually to implement it.

Static verification allows you to check if a model has some desired “correctness” properties like tracing (the occurrence of a specific series of activities, consequence (if some activities or events lead to the execution of other activities or reaching of specific goals), a.o. Reference [13] lists some useful execution patterns for business processes from the business perspective. An extensive discussion on the application of model checking techniques for the verification of qualitative dynamic properties of systems specified with FSP can be found in [9].

Dynamic simulation allows the estimation of performance indicators of a business process: degree of utilization of resources, cycle times, waiting times, a.o. ([17]).

6. Related work

Some work on mapping RAD to simulation models of discrete event systems has been reported in [17]. The difference between our work and [17] is that there, the target is deriving a simulation model suitable for quantitative analysis only, while in our approach the target is a more abstract process algebra model that is suitable for both qualitative and quantitative analysis.

The use of process algebras for providing a formal semantics of business processes is not entirely new ([23], [6], [22], [1], [14]). Paper [23] uses the Process Interchange Format (PIF hereafter, [15]) as the starting point for formal verification of qualitative dynamic properties of business processes. PIF is a textual notation devised for exchanging business process models between various software tools. The main result reported in [23] is an algorithm for mapping a PIF model to a process algebra model specified in CCS ([18]).

Paper [6] describes a first attempt to provide a formal semantics to RAD models via process algebras. The authors use PEPA, a process algebra specially tailored for devising performance profiles of discrete event systems ([10]). However, not all the aspects of RAD models are adequately addressed, in particular the modelling of part refinements that return to a common thread has some problems. The main difference of the approach taken in the present paper is mapping a role model to a composition of state models such that the synchronization of the events that produce/consume tokens ensures the right ordering of activities, while in [6] this is ensured by the use of the prefix operator. A similarity between the two approaches is the fact that a RAD model is mapped to a parallel composition of role models, i.e. a role model corresponds roughly to a component in PEPA terminology.

The emergence of UML ([2]) as a recent standard for modelling software and business systems has catalyzed a lot of research on providing a formal semantics to the various diagrammatic notations incorporated into the UML. In particular, this research has also addressed UML Activity Diagrams (AD hereafter), the UML notation specially devised for workflow and business process modelling. Within this context, at least two papers – [22] and [1] that address the problem of proposing a formal semantics to AD via process algebras can be cited. Note that even it is not the goal of this paper to provide a comparison of RAD and AD, it worth mentioning here the work reported in [22] and [1] because clearly, a lot of similarities exist between RAD and AD including: i) focus on activities and states; ii) separation of roles (swimlanes in AD); iii) usage of control structures (refinements in RAD and splits and joins in AD).

Paper [22] describes a mapping from AD to FSP with the goal to derive a model of computation expressed in LTS so as to be able to check the behaviour expressed in the AD specification ([22]). The approach from [22] has similarities with both the approach described in [6] and the approach described in the present paper. At the level of a role model the approach in [22] it is more closer to [6] with the difference that it addresses correctly the modelling of and joins and thus the level of granularity of using parallel composition inside a role model is slightly lower than in [6] (but clearly higher than in the approach discussed in the present paper). The overall behaviour expressed in an AD is modeled in [22] as a parallel composition of models of individual AD and swimlanes, similarly with our approach.

Paper [1] provides a formal semantics for AD by describing them in terms of the process language CSP ([11]). Using a result that states that a class description of a system can be expressed as an abstract data type and then translated to its corresponding CSP process, one can check if the final class description is consistent with its AD specification. The approach described in [1] for mapping an individual AD to a CSP model resembles closely the approach discussed in the present paper. States are mapped individually to CSP processes and the model of an AD is the parallel composition of the resulting processes. The synchronization of the line events ensures that the actions occur in the correct order. There is a notable difference, however. [1] uses the interleaving parallel combination operator of CSP in a recursive loop to enable the incoming line events to occur many times before a state transition occurs, thus yielding an infinite state process, while in our approach the resulting
process is finite state. Another difference is that [1] doesn’t address systems composed of multiple AD and swimlanes.

FSP has already been proposed as a mechanism for modelling and analysis of workflow processes in [14]. There, a workflow schema is defined as a collection of tasks and their notification and/or dataflow dependencies. Then mappings to FSP are defined for task interfaces, primitive and composite tasks. The main difference between this approach and our approach is the formalism for the definition of the workflow schema which has a functional flavour resembling IDEF-based methods for business process modelling ([5]) rather than state-based, as RAD or AD.

7. Conclusions

In this paper we have shown that the RAD notation for business process modelling has a natural mapping to the FSP process algebra. The mapping was presented through a translation algorithm of a RAD model to a FSP model. The resulting FSP model can be used to derive a LTS computational model of a business process that is useful for checking the behaviour of the business process before actually having to implement it in practice.

References