From Process Algebra to Visual Language

Antonio Cerone

Software Verification Research Centre
The University of Queensland, QLD 5001,
Email: antonio@svrc.uq.edu.au

Abstract

The syntactical constructs and the features of three well-known process algebras, CCS, CSP and Circal, are analysed and compared. The analysis is carried out from the point of view of the system designer and aims to single out which features make the modelling process easier, facilitate the verification phase, provide a better intuition of the system behaviour, and are more appropriate for the visualisation of both the functional structure of the overall system and the behaviours of the components in which it has been decomposed. The results of such an analysis are then exploited to propose a visual framework for the design and verification of systems, which is targeted to users who are not necessarily expert in formal methods.

Keywords: Process algebra, visual notation, verification.

1 Introduction

In the last decade computers have been widely used to control safety-critical systems such as physical processes, mechanical devices, transportation and communication networks. The complexity of such systems and the need to meet strict safety requirements has motivated the development of formal specification languages that allow rigorous modelling and the application of formal verification techniques (Peled 2001). The use of such languages and the associated verification techniques often requires a deep knowledge of the underlying mathematical theory.

The designers of a computer system are seldom expert in formal methods and have more often an engineering rather than mathematical background. They are more familiar with visual and tabular representations, such as behaviour tables, state diagrams, timing diagrams, flow charts, Petri nets and statecharts. Among these visual notations only Petri nets have a universally accepted underlying semantics and are associated with formal analysis techniques (Reisig 1985). Therefore, only Petri nets can be considered formal methods. All other visual notations need to be given a formal semantics, which transparently encapsulates formal methods and its associated analysis techniques within a framework which may easily be used by an average designer. Statecharts have a rigorous visual syntax (Hard 1987), but no universally accepted underlying semantics. Attempts have been made to retrofit suitable semantics to the visual syntax (von der Beeck 1994), resulting in more than one hundred alternative semantics, none of which is fully satisfactory.

By contrast to the Statechart approach we aim to build a visual framework inspired by widely used formal specification languages. In particular we analyse three well-known process algebras, the Calculus of Communicating Systems (CCS) (Milner 1989), Communicating Sequential Processes (CSP) (Hoare 1985) and the Circal process algebra (Milne 1994). In our analysis we try to single out the features of different process algebras that make the modelling process easier, facilitate the verification phase and provide a better intuition of the system behaviour. Such features are then utilised to define a visual framework, which we call Visual Process Algebra (VPA).

In Section 2 we present an overview of CCS, CSP and Circal. Section 3 critically analyses the features provided by the three process algebras and the way such features may be exploited in modelling and verifying real-life systems. Section 4 discusses possible approaches to visually represent non-determinism, communication, behaviour and system structure, and summarizes related work. Finally, Section 5 defines our proposed Visual Process Algebra.

2 Modelling with Process Algebras

Process algebras are mathematical formalisms for modelling the behaviours of systems in a structured way. In this section we give an overview of three among the most popular process algebras: CCS (Milner 1989), CSP (Hoare 1985, Roscoe 1994) and Circal (Milne 1994). We have selected these three process algebras which cover most of the features provided in the process algebra world. Some widely-used process algebras have been left out, such as LOTOS (Bolognesi & Brinksma 1987), whose features are basically the same as CSP’s, and the Algebra of Communicating Processes (ACP) (Beaten & Weijland 1990), whose extension with silent action and abstraction (ACP∗) is very interesting from a theoretical point of view, but seldom applied to practical modelling.

The syntax of the three selected process algebras is summarized in Table 1. The basic entities of a process algebra are:

- processes which are the units of behaviour;
- actions which define communications among processes.

Every process has a sort, which defines the set of actions used to communicate with other processes.

The simplest process is the termination process, which has no behavior. It is denoted by 0 in CCS, by STOP in CSP and by Δ in Circal. Every process can be guarded by an action in CCS and CSP and by a set of actions in Circal.

\[
\begin{align*}
\text{(CCS)} & : a \cdot P \\
\text{(CSP)} & : a \rightarrow P \\
\text{(Circal)} & : (a) P
\end{align*}
\]
The above notation denotes a process that performs action \( a \) and then behaves as process \( P \). In Circal a process can also be guarded by a set of more than one action, say \( \{a, b, c\}\). A process \( P \) guarded by a set of actions \( \{a, b, c\}\) is expressed in Circal as \( (a \; b \; c) \; P \). When the set consists of just one action, as in \( \{a\} \; P \), the shorthand \( a \; P \) can also be used. In the semantics of Circal the actions that belong to a set which guards a process must always occur simultaneously. Therefore, by contrast to CCS and CSP where concurrency is always represented as non-deterministic interleaving, Circal allows an explicit representation of true concurrency.

Processes can be given names using the definition operator. In this paper we will use \( \triangleq \) to denote the definition operator in all three process algebras. The definition operator allows recursive definitions as in the following examples.

\[
P \triangleq a.P \quad P \triangleq a \rightarrow P \quad P \triangleq a \; P \quad P \triangleq a \; P \\
\text{(CCS)} \quad \text{(CSP)} \quad \text{(CircaL)}
\]

In CCS and CSP recursive definitions are also given using a fixpoint operator, but this is outside the scope of our analysis.

2.1 Choice Operators: Determinism versus Non-determinism

Process algebras provide mechanisms for describing behaviours which present choices of actions. A crucial point is the distinction among deterministic and non-deterministic choice. In CCS there is only one choice operator ("\( \rightarrow \)") and non-determinism is represented either by having distinct alternatives starting with the same action or by having distinct alternatives starting with the \( \tau \) silent action.

CSP has instead three different choice operators:

- **guarded alternatives** the \( | \) operator allows deterministic choice, made by the environment, between alternatives with distinct guards;
- **non-deterministic choice** the \( \sqcap \) operator allows non-deterministic choice, made autonomously by the process, between distinct alternatives.
- **external choice** the \( \Box \) operator allows a choice, made by the environment when the guards are distinct, between distinct alternatives;

Notice that the external choice operator behaves as the guarded alternatives operator when the alternatives have distinct guards, whereas it behaves as the non-deterministic choice operator when the alternatives have identical guards.

CircaL has two operators, the external choice \( + \), which corresponds to the CSP \( \Box \) operator, and the internal choice \( \& \), which corresponds to the CSP \( \sqcap \) operator. These CircaL operators have the same semantics as the corresponding CSP operators.

These definitions have their graphical counterpart in Figure 2. A such a figure is also the graphical counterpart of a third possible CSP or CircaL textual definition of the same behaviour, in which the external choice in \( P \) between the two terms guarded by \( a \) is replaced by a non-deterministic choice.
2.2 Communication and Concurrency

Communication between concurrent processes is expressed by defining

1. the way actions of distinct processes may interact;

2. which processes are involved in the interaction.

The way actions interact is the key point in defining communication and is also the main difference among the three process algebras we are analysing.

In CCS actions are partitioned into two complementary classes: input and output actions. There is a bijection between input and output actions, which defines the pairs of complementary actions. An input action is denoted by a lower case letter and its complementary output action is denoted by the barred version of the same letter. For example, input action $a$ and output action $\bar{a}$ are complementary. In CCS communication may only occur through the synchronisation of complementary actions that belong to the intersection of the sorts of distinct processes. Therefore, CCS only allows two-party communication. Moreover, when two complementary actions synchronise, neither of them appears in the system behaviour, but the $\tau$ silent action is performed instead.

In CSP and Circal synchronisation may occur among actions with the same name. Since an action may belong to the sort of an arbitrary number of processes, there is no limit to the number of processes that can communicate by synchronising on the same action. CSP and Circal, therefore, allow multi-party communication.

In each of the three process algebras there is an operator of parallel composition which selects the processes involved in the interaction. Such an operator is denoted by $|$ in CCS, by $\parallel$ in CSP, and by $*$ in Circal. The parallel composition operator is often used in combination with the restriction operator, denoted by $\backslash$ in CCS, and with the hiding operator, denoted by $\bar{\ }$ in CSP and by $- \in$ in Circal, in order to prevent other processes from participating in the interaction.

The restriction operator of CSP forces the actions contained in the set to which the process is restricted to synchronise with the complementary action. For example, process

\[(a.P + b.Q) | (\bar{a}.R)\backslash\{a\}\]

can perform either $\tau$, as a result of the synchronisation of $a$ and $\bar{a}$, or $b$, but cannot perform either $a$ nor $\bar{a}$ in isolation, which are in fact forced to synchronise by the restriction. Unrestricted process

\[(a.P + b.Q) | (\bar{a}.R)\]

can instead perform $\tau$, as a result of the synchronisation, or $a$ or $\bar{a}$ or $b$.

The hiding operator of CSP and Circal has a much simpler role than the restriction operator of CCS. It just hides the occurrences of some actions. For example, CSP process

\[((a \rightarrow P + b \rightarrow Q) \parallel (a \rightarrow R))\backslash\{a\}\]

which is equivalent to Circal process

\[((a + b.Q) * (a.R)) - a,\]

can perform $b$, but not $a$. This does not mean that the synchronisation between the two $a$ actions on the two sides of the parallel composition operator does not occur, but that such a synchronisation is hidden to an external observer.

Parallel composition operators introduce concurrency in process algebras. However, the parallel composition operator is not primitive. Every process algebra has an expansion law which redefines a process which expresses concurrency without the use of the parallel composition operator.

In CCS and CSP concurrency is defined as a choice of all possible interleavings of the actions offered by the components. For example, CCS process

\[a.0 | b.0\]

is equivalent to process

\[a.b.0 + b.a.0\]

In Circal, instead, actions can also occur simultaneously. Therefore, concurrency is defined as a choice which includes both of all possible interleavings and all possible simultaneous occurrences of the actions offered by the components. For example, Circal process

\[(a \Delta) * (b \Delta)\]

is equivalent to process

\[a.b \Delta + a \Delta + (a b) \Delta\]

The last operator in Table 1 is the renaming or relabelling operator. Function $f$ and relation $R$ are both expressed as a sequence of replacements of single actions. For example, $f = [c/a, d/b]$ denotes the replacement of $a$ with $c$ and $b$ with $d$. In CCS $f$ must be such that $f(x) = f(\bar{x})$ for each action $x$, and $f(\tau) = \tau$. If $f$ is not injective, then the renaming may introduce non-determinism. For example, the renaming

\[(a.P + b.Q)[c/a, c/b]\]

generates process

\[c.P + c.Q\]

In CSP the renaming by a relation, called relational renaming, may introduce non-deterministic choices with distinct guards. For example, the renaming

\[(a \rightarrow P \square b \rightarrow Q)[c/a, d/a]\]

generates process

\[(c \rightarrow P \square d \rightarrow P) \square b \rightarrow Q\].

In Circal renaming can be defined in terms of the parallel composition and hiding operators, by exploiting the simultaneity of actions. For example, the renaming

\[(a.P + b.Q)[c/a, d/b]\]

can be also achieved by the expression

\[(a + b.Q) * R_{\{a,c\}} * R_{\{b,d\}} - a b,\]

where $R_{\{a,c\}} \triangleq (a c) R_{\{a,c\}}$ and $R_{\{b,d\}} \triangleq (b d) R_{\{b,d\}}$.
3 Learning from Experience

In this section we critically analyse the features provided by the three process algebras and the way such features may be exploited in modelling and verifying real-life systems. Our analysis aims to single out those features that are closer to the designer's view and that can be more easily expressed in a visual framework.

3.1 Communication

From the specifier's point of view, the use of a two-party synchronisation between complementary actions, as in CCS, allows a more realistic representation of a directed communication, in which one of the processes produces an output and sends it to another process, which observes it as an input. The direction of the communication is, however, a pure convention, since the essence of the behaviour of the system does not change if we swap input and output actions. The mechanism that forces synchronisation using the restriction operator is not intuitive from the point of view of the designer and cannot be easily expressed in a visual representation.

The use of a multi-party communication, as in CSP and Circal, allows the broadcast of information to several receivers. In this case the distinction between input and output is in general even more vague than in CCS two-party communication. Non-distinguishing between input and output is a disadvantage for the specifier, who cannot easily see the direction of the data-flow within the system, but it may be an advantage for the verifier, who can simulate a system using a test pattern containing "output" actions mixed with "input" actions.

A common problem in process algebras is that when a process provides the same service to several identical client processes, it has to use for each single client a distinct action for accepting service requests from that client. This problem may be solved with the use of an indexed choice, but in a visual representation we would still need to explicitly represent one distinct communication channel for each client process.

3.2 Constraint-based modelling

An important modelling methodology that is available in process algebras which provide synchronisation among an arbitrary number of processes, such as CSP and Circal, is the constraint-based modelling methodology (Vissers, Scoo, van Sinderen & Brinkman 1991). When a process P is composed with a process Q, we say that Q constrains P if and only if there is a part of the behaviour of P whose restriction to the intersections of the sorts of P and Q is not consistent with the behaviour of Q. It is therefore possible to define constraints in terms of processes. For example, CSP process

\[ Q' \triangleq a \rightarrow b \rightarrow Q' \]

constrains process P given in Figure 1 because in P || Q' action b is prevented from being performed initially (a must be performed before b in Q'), whereas CSP process

\[ Q'' \triangleq a \rightarrow c \rightarrow Q'' \]

does not constrain P because the restriction of P || Q'' to \( \{a, c\} \) is equivalent to Q'' itself (and is therefore consistent with the behaviour of Q'').

The constraint-based modelling methodology has been widely exploited in CSP and Circal to characterise and verify system properties using a model-checking approach (Cerone, Cowie, Milne & Moseley 1997, Cerone, Kearney & Milne 1998, Cerone & Milne 2000, Roscoe 1994). The constraint-based modelling methodology supports a clear characterisation of safety properties. Let P be a process of sort \( \mathcal{L}_P \) and Q be a process of sort \( \mathcal{L}_Q \subseteq \mathcal{L}_P \). Let us suppose that Q models a safety property which might or might not hold in the system modeled by P. If Q constrains P, then the property represented by Q is not implicitly modelled in P; on the other hand if Q does not constrain P, then the property represented by Q is implicitly modelled in P, that is, the system satisfies the property. Therefore the verification methodology consists of checking whether or not the process Q that represents the safety property to be verified constrains the process P that represents the system. This can be expressed in CCS by the equivalence P || Q ≅ P. For example, for processes P, Q' and Q'' above, equivalence P || Q' ≅ P is false whereas equivalence P || Q'' ≅ P is true. Process Q' models the safety property that b must always be preceded by a and process Q'' models the safety property that c must always be preceded by a. We can notice in Figure 1 that the first safety property does not hold in P whereas the second safety property holds in P.'

The fact that the constraint-based modelling methodology, which is so important both in design and verification, is possible only within a framework which supports multi-party communication is a strong argument in favour of this form of communication.

3.3 Simultaneity of actions

An advantage of Circal with respect to all other process algebras is the fact that it allows the simultaneous occurrence of different actions. Such a feature has been exploited in modelling hardware at the gate level (Milne 1994) and the transistor-level (Cerone & Milne 1999), in modelling true concurrency and causality (Cerone 2002), in representing priorities among actions (Cerone, Cowie, Milne & Moseley 1996, Cerone et al. 1997), in modelling dense time (Cerone & Milne 1997) and in the verification of performance properties (Cerone et al. 1998, Cerone & Milne 2000). This is an evidence that the simultaneity of actions enriches the ability to model aspects of concurrent systems which are believed hard to characterise in process algebra frameworks.

4 Textual versus Visual Representation

There are a few issues that must be solved in moving from a process-algebraic to a visual representation. Two crucial points are the visualisation of non-determinism and communication.

4.1 Non-determinism

If we look at Figures 1 and 2 it is not immediate to understand that the two graphical representations describe the same behaviour. The graphical representation in Figure 3 could also appear a third representation of the behaviour given in Figure 1. However, in Figure 3 the unlabelled transition shows that the process can autonomously decide to perform a, thus refusing an offer of b by the environment.

These examples show that the use of unlabelled arcs (or, equivalently, arcs labelled by a special action, such as CCS's \( r \) action) and of choices with identical
4.2 Communication

In process algebras communication channels are defined by the relationships among the names of actions that belong to the sorts of the processes that are composed together using a parallel composition operator, as shown in Section 2.2. In CCS two actions establish a channel if they have complementary names; in CSP and Ciral if they have the same name. However, relations among names, such as identity or complementarity, are not easy to visualise. We cannot expect the analyst of a graphical design to try to match occurrences of the same name, which are spread throughout the design. The eyes of the analyst need to be driven by lines connecting the entities involved in the communication. The most natural way of representing communication in a visual framework is therefore the use of communication lines connecting communication ports. This approach is used by several visual notations for process algebras (Boudol, Roy, de Simone & Vergamini 1989, Roy & de Simone 1990, Milner 1994). In such a context names lose their role in defining the communication and become pure annotations, whose purpose is to make explicit the interpretation of the design and increase readability.

4.3 Behaviour versus Structure

We have seen in Section 2.2 that the parallel composition of two processes can be expanded into a non-deterministic interleaving (plus all possible simultaneous occurrences when modelling in Ciral) of the actions performed by the two processes. This means that any process can be expanded into a global behaviour through iterated applications of the expansion law.

Behaviours can be visualised using the classical graphical representation for finite state machines as shown in Figure 1. Therefore, in principle every system could be visualised through the graphical representation of its global behaviour. Unfortunately, such a global behaviour is usually too large to be represented as a whole and no longer carries information about the system structure described by the original design. Such a system structure has been defined by the designers and represents the natural decomposition in which the system expresses its features and functionalities.

In a process algebra framework a system can be modelled using any combination of choice and parallel composition. This leads to a very complex model in which the behaviour and the structure of the system are heavily intertwined. Such a complex model does not provide much intuition of the system behaviour.

4.4 Related work

One of the first and most successful attempts for embedding process algebras within a visual framework is Autograph (Boudol et al. 1989, Roy & de Simone 1990), one of the tools developed within the Meije Project (Madalaine 1995). Autograph is a graphical editor for pictures that are later interpreted into process algebra terms. Autograph is based on two main types of editable objects, networks and automata, which allows the separate definition of structure (as networks) and behaviour (as automata) that we have described in Section 4.3. However, the graphical representation can then be interpreted into different process algebras. The representation of non-determinism is just a graphical counter-part of the syntax of the process algebra used for the interpretation. Therefore, Autograph suffers from the modelling problems described in Section 4.1.

Milner has defined the \( \pi \)-nets (Milner 1994), a visual notation close to the \( \pi \)-calculus, which has later evolved into faithful visual representations of the \( \pi \)-calculus (Milner 1996, Gobrun & Rotaru 1998). However, these works do not aim at the definition of languages for visual design, but present visual notations as auxiliary tools to be used in theoretical investigation (Milner 1994).

Petri nets have an underlying formal semantics and are associated with formal analysis techniques (Reisig 1985). Moreover, the distributed representation of the state of the overall system allows the visualisation of the system behaviour through a token game animation of the design. However, it is not likely, at least during the initial phases of the design, to have a clear mental model of the evolution of the distributed state of the system. The system model is usually built either by refining a coarse global mental model into more and more detailed designs, i.e. using
a top-down approach, or putting together the designs of very small components for which it is easy to have a mental model, i.e. using a bottom-up approach.

The Box Calculus (Best, Devillers & Hall 1992), an algebraic representation of Petri nets that supports composition and decomposition, is the first attempt to allow the use of Petri nets in top-down and bottom-up modelling approaches. The calculus has then been used as a base for defining the $B(PN)^2$ imperative programming language (Best & Hopkins 1993), which allows the expression of concurrent algorithms, and the PBC process algebra (Best & Koutny 1995), which is based on CCS. Both $B(PN)^2$ and PBC have been implemented by the PEP tool (Grahmann & Best 1996).

Statecharts (Harel 1987) are an attempt to visualise a system consisting of components which are combined together through an intertwined mix of choices and concurrency. The result is a graphical design decomposed into OR-states and AND-states. Each OR-state consists of a choice of other OR-states and AND-states, whereas each AND-state consists of a parallel composition of other OR-states and AND-states. However, such a visual design gives very little intuition about the behaviour of the overall system. Even worse, Statecharts do not build on a consolidated semantic base. Many distinct semantics have been suggested for Statecharts, but none of them have been universally accepted (von der Beeck 1994). Moreover, even if we accept one of these alternative semantics, a person who tries to analyse the model has to change reasoning context every time he or she moves from an OR-state to an AND-state and vice versa.

5 Towards a Visual Process Algebra

In this section we exploit the results of the analysis of the features of CCS, CSP and Circal carried out in Sections 3 and 4. We define a visual framework which incorporates the features that make the modelling process easier, facilitate the verification phase and provide a better intuition of the system behaviour.

5.1 Making Non-determinism Implicit

We have said in Section 3.1 that having a clear distinction between input and output is convenient from the point of view of the specifier. In our proposed visual notation we want to have such a distinction within a multi-party communication paradigm in order to allow a more flexible form of communication, which can be used within the constraint-based modelling approach presented in Section 3.2.

We give to input and output actions an interpretation which allows us to avoid as much as possible the explicit representation of non-determinism. An output action is performed by the process independently of the offer by the environment. An input action is instead the acceptance of a stimulus from the environment. Therefore we can say that output actions implicitly carry non-determinism.

Let us assume the convention of CCS of representing output actions as barred letters. The process given in Figure 1 can be represented using input and output actions, which make the non-determinism implicit as shown in Figure 3. The interpretation of such a behaviour is the following: the process idles while the stimulus offered by the environment is $b$, until it accepts a stimulus $a$ from the environment, which triggers a non-deterministic choice between two possible output actions, $\tilde{a}$ and $\tilde{b}$. Notice that from the environment's perspective stimuli $a$ and $b$ are actually output actions $\tilde{a}$ and $\tilde{b}$. If the environment potentially offers both $\tilde{a}$ and $\tilde{b}$, there is also a non-deterministic choice driven by the environment, which forces process in Figure 5 to accept either $a$ or $b$. The choice between $\tilde{a}$ and $\tilde{b}$ is internal to the environment and is therefore invisible to the process in Figure 5.

If we use actions $c$ and $d$ in place of $\tilde{c}$ and $\tilde{d}$, respectively, as shown in Figure 6, then we obtain a different behaviour with respect to the one in Figure 1. The choice between input actions $c$ and $d$ is now driven by the environment, through an environmental choice (internal to the environment) between $\tilde{c}$ and $\tilde{d}$. Therefore, the process in Figure 6 is fully deterministic.

The process given in Figure 3 cannot be modelled using input and output actions to make the non-determinism implicit. In this case the non-determinism is not a simple choice of actions, but a non-deterministic choice between action $a$ and the deterministic choice between action $a$ and action $b$. This is expressed in CSP notation by

$$a \rightarrow P_1 \cap (a \rightarrow P'_1 \cap b \rightarrow P)$$

Such a form of non-determinism cannot be implicitly encapsulated within output actions, but requires the use of unlabelled transitions. In general, a designer never needs to directly use such a form of non-determinism, and therefore will never use unlabelled transitions, when modelling the behaviours of the system components. However, non-determinism may be generated by hiding some actions in the component design. For example, the behaviour given in Figure 3 can be obtained from the behaviour in Figure 7 by hiding actions $x$ and $y$.

5.2 Behaviour Visualisation

In this section we finally move from process algebraic notations to a visual notation, which we call Visual Process Algebra (VPA). A behavioural process is represented in VPA by a box which shows on the outline the communication ports (which define the sort of the process) and contains the representation of the actual behaviour. A communication port is visualised by icon $\circ$, if it defines an input action, and by icon $\bullet$, if it defines an output action.
VPA adopts the Circal paradigm of guarding processes with sets of actions. The occurrence of a set of actions is called an event. The actual behaviour of a process is visualised as a state-transition graph as follows:

- A state is visualised by a circle.
- An event is visualised by icon •, if it contains the occurrence of at least one input action (agreed events), by icon ◦, otherwise (output events).
- For each action occurring in an event, the icon of the event is connected through a dotted line to the icon of the communication port that represents that action.
- A transition between states is visualised by two arrows, one from the source state to the icon of the event that labels the transition and the other from the icon of the event to the target state.

Notice that the null event, which is the occurrence of an empty set of actions, is an output event. This is consistent with the fact that the null event represents the kind of non-determinism that cannot be defined by output actions. In general a null event is never used by a designer but may result as a consequence of the hiding of some action as in the behaviours given in Figures 3 and 7.

Let us see the following example. The VPA behavioural process given in Figure 8 describes a vending machine which sells tea and cakes. When the machine is in state Rdy, ready to accept a request, the customer may select "tea" (input req.) or "cake" (input req.) and the machine instantaneously displays the price of a tea (output price_t), and changes to state R_t, or the price of a cake (output price_c), and changes to state R_c, respectively. A customer who wishes to purchase both tea and cake may select "cake" from state R_t or "tea" from state R_c and the machine instantaneously displays the sum of the prices of a tea and a cake (output price_t-c), and changes to state R_t-c. In each of the states R_t, R_c, and R_t-c, the machine is waiting to accept the right payment (inputs pay_t, pay_c and pay_t-c, respectively) which will trigger a change to state P_t, P_c, and P_t-c, respectively, from where the requested products are delivered (output tea, simultaneous outputs tea and cake, and output cake, respectively). After delivering the products the machine is in state Del, from which it goes back to the initial state (output ready).

Output events consist of only output actions and are, therefore, a potential source of non-determinism. The vending machine in Figure 8 is, however, fully deterministic because each output event is the only possible choice from its source state.

Agreed events contain at least one input action, which forces the process to look for an agreement with another process before the event can occur. For example, in state Rdy the choice between event \{req, price_t\} and event \{req, price_c\} depends on which input req. or req. is provided by the environment, which in this context is the customer.

The set of actions that define an event can be explicitly represented with the following notation, which emphasises the roles of actions as inputs and outputs: the list of the input actions, separated by commas, followed by symbol \(\uparrow\), followed by the list of the output actions, separated by commas. The null (output) event is represented by just \(\uparrow\).

Figure 9 shows the same vending machine as in Figure 8 visualised with the dotted lines connecting communication ports and events replaced with an explicit labelling of the events. In a potential tool based on VPA, a user might choose between the visualisation in Figure 8 and the one in Figure 9, or even to have both dotted lines (which in a tool could be lines of a different colour) and explicit representation of events.

5.3 Structure Visualisation

VPA adopts the hierarchical decomposition of a system represented in Figure 4. We have seen in Section 5.2 how to visualise the behavioural processes, which are the leaves of such a hierarchy. We have also said in Section 4.2 that the most natural way of visualising communication is the use of communication lines among ports.

In VPA we would like to distinguish between two kinds of communication channels, a dynamic channel, visualised by icon ( ), and a broadcast channel visualised by icon ( ). The icon of a channel is connected through arrows, which highlight the direction of the communication, to all ports involved in the communication represented by that channel. System components and icons of communication channels are enclosed in a box whose outline shows the communication ports visible to an external observer. In order to make a communication visible to an external observer, the icon of the communication channel needs to be connected with a line to exactly one port on the outline of the composite system. If such a port is an output port we say that the communication is visible as an output. If the icon of the communication channel is not connected to any port on the outline of the composite system we say that the communication is invisible.

A dynamic channel represents a communication involving any number greater than 1 of actions. However, at any time no more than one input action and one output action define the communication. A dynamic channel consisting of only input actions is called an input choice and a dynamic channel consisting of only output actions is called an output choice. Input choices can be either visible as inputs or invisible. Dynamic channels which are not input choices can be either visible as outputs or invisible.

A broadcast channel represents a communication involving any number greater than 1 of actions. At any time any number of ports may participate in the communication, which may cause collision on the channel. Collisions may be avoided by filtering multiple outputs to the broadcast channel through an additional dynamic channel. A broadcast channel consisting of only input actions is called a synchronisation channel. Synchronisation channels can be either visible as inputs or invisible. Broadcast channels which are not synchronisation events can be either visible as outputs or invisible.

The use of a dynamic channel overcomes the problem described in the last paragraph of Section 3.1. In VPA the request for a service received by a process can be modelled by an input port connected through
Figure 8: VPA model of a vending machine

Figure 9: VPA model of a vending machine with transition annotation
the same dynamic channel to the output ports modelling that request, one for each client.

Let us suppose we have a system consisting of the vending machine in Figure 8 and two customers \( C_1 \) and \( C_2 \), each purchasing just tea, and that we are interested in making visible only actions \( \text{req}, \text{tea} \). The structure of such a system is described in VPA as shown in Figure 10. We have named channels after the action of the vending machine which is involved in the communication on that channel. Channel \( \text{ready} \) is a broadcast channel from the vending machine to the customers. We can interpret this action as the vending machine displaying a message ready, which can be read by all customers. Channels \( \text{req}, \text{price}, \text{pay}, \text{tea} \) are dynamic channels, which can connect only one of the two customers at a time with the vending machine.

The two customers have the same sort, but their behaviours can be different. We can even suppose a scenario where customer \( C_1 \) requests a tea and pays, but when the tea is delivered \( C_2 \) steals it. Such a scenario is possible due to the use of dynamic channels: the communication on \( \text{req} \) may occur between \( C_1 \) and the vending machine, whereas the successive communication on \( \text{tea} \) may occur between the vending machine and \( C_2 \). Such a scenario is implicit in the VPA model of the vending machine. Using instead a process algebra to model the vending machine such a scenario needs to be explicitly represented. A model that does not take such a scenario into account does not reflect reality correctly but may contain unwanted assumptions on the environment. In a safety-critical system such unwanted assumptions might lead to the dangerous conclusion that the system is safe when in reality it is not safe at all. In our example the unwanted assumption would be that no customer can steal the tea requested and paid for by another customer. This may lead to the wrong conclusion that the vending machine protects customers from theft.

6 Conclusion and Future Work

After a critical analysis of three well-known process algebras and their use in modelling computer systems, we have proposed a visual framework for system design which is built on the results of such analysis. Our visual framework, which we call Visual Process Algebra (VPA), addresses in a visual fashion those features of process algebras which have proven to ease modelling and better support formal verification.

VPA allows an implicit representation of non-determinism in terms of input and output actions. Communication is based on two kinds of channels, dynamic and broadcast channels, which allow both two-party and multi-party communication. These two forms of communication channels are very reliable in preventing the designer from implicitly modelling unwanted assumptions. Moreover, VPA supports the constraint-based methodology and the verification technique that we have presented in Section 3.2.

Our presentation of VPA is quite informal and is open to discussion and criticisms. The visual syntax still needs to be given a formal semantics. Having based the construction of our visual language on the features of existing process algebras should facilitate the definition of a formal semantics, which is part of our future work.

Acknowledgements

I would like to thank Colin Fidge for reviewing a draft of this paper providing many useful comments. Many thanks also to the anonymous reviewers who constructively criticized the paper greatly contributing to its improvement.

References


