

Diagram Models for Interacting Components

Artur Zawłocki^{1,2}

Uniwersytet Warszawski, Instytut Informatyki, Warszawa, Poland

Abstract

We present a semantic model and a logic for systems of concurrent components. Following the categorical approach, we define a category of component models in which limits can be used to construct systems from simpler components. A novel idea is to use diagrams in this category, not just limit objects, as models for the logic. The resulting “diagram logic” allows one to specify both behavioural and structural aspects of systems: temporal operators and structural predicates can be freely interleaved. As in first-order logic, system components can be quantified over, bound to variables and classified by predicates. There is a price we must pay for the expressive power of the logic: the set of valid formulae is not recursively enumerable.

Keywords: categorical models, component interaction, logic

1 Introduction

We consider systems composed of a number of *components* running concurrently and coordinating their activities. Examples of such systems include an operating system running multiple processes, a web server interacting with many clients, a distributed application executing in a mobile telephony network, and also a multitude of non-software, “real-world” systems. As should be clear from the above examples, we use the term “component” to denote “run-time” entities, like processes or active objects.

A well-established and elegant mathematical approach is to use category theory to model concurrent components and their interaction. In particular, the inspiration for this paper is the work of Nielsen and Winskel on categorical models of concurrency. In [16] they have related several models of concurrency, given as categories in which an object c represents a *component*, and a morphism $f : c \rightarrow d$ identifies d as a *subcomponent* of c . Under such interpretation, a categorical diagram represents a system of components, interacting by sharing common subcomponents. A limit of a diagram, if it exists, allows one to represent the whole system as a single component, thus disregarding its internal structure while retaining the external behaviour.

¹ Email: zawlocki@mimuw.edu.pl

² This work was supported by the EU funded IST Project SENSORIA.

A category of component models—similar to labelled transition systems of [16]—is introduced in Section 2. We use it to illustrate the *system-as-diagram* principle. We equip our categorical model with a temporal logic in which behaviour of components can be specified.

Several specification formalisms has been based on the idea of using categorical (co)limits to represent systems built from interacting components. The idea can be traced back to Goguen’s work on General Systems Theory in early seventies (cf. e.g. [9]). It also underlies sheaf semantics of concurrent objects ([15], [8]).

In similar spirit, Fiadeiro and Maibaum considered in [7] a category of theories of temporal logic, in which a theory describing a system can be obtained as a colimit of theories for its subcomponents. This idea was explored further in the architectural description language CommUnity ([6]). Similar approach is taken in [13] and [15].

The advantage of categorical formalisms is their compositionality. The drawback is that they describe systems built from a fixed number of components, and interconnected in a static way. System specification has usually two levels: one describes component interconnection using a simple formal language or a graphical notation, and another one describes temporal behaviour of the components in a temporal logic. From these data, the joint behaviour of the system can be inferred.

In this paper we propose a logic that allows one to freely interleave the two aspects of system description: the structural and the behavioural one. This is achieved by evaluating formulae in categorical diagrams, not just their limits. The logic has a first-order flavour: diagram nodes can be quantified over with component variables, there is a built-in component equality and a predicate stating that one component is a part of another one. Using diagrams as models, the logic can describe behaviour of individual components as well as their possible configurations.

The logic for diagrams is presented in Section 3 as an extension of the component logic of Section 2.

An small example system specification is given in Section 4. Section 5 contains some results regarding the expressive power of the logic. Most notably, we remark that the validity problem is not recursively enumerable, as in case of standard first-order temporal logics.

Finally, Section 6 contains a short summary and reports some related work.

Acknowledgements

The author is grateful to Grzegorz Marczyński, Andrzej Tarlecki and three anonymous referees for valuable comments and suggestions.

2 Component Models and Component Logic

Our model of component execution is based on familiar *labelled transition systems* (cf. [16]). A transition system model of a component consists of a set, whose elements represent the component’s states, and a relation representing the possible state transitions. Each transition is labelled with an action symbol, meaning that the transition is caused by an execution of the action. Additionally, we equip transition systems with state-dependent attributes, with values in a fixed data algebra.

2.1 Component Transition Systems

For the rest of this paper we assume a fixed algebraic many-sorted *data signature* Δ with the set of sorts $\text{srt}(\Delta)$, and a fixed Δ -algebra D of *data values*.

Definition 2.1 A *component transition system* $\mathbb{T} = \langle A, V, S, T, \text{lab}, \text{val} \rangle$ consists of a set A of *actions*, a set V of *attributes*, a set S of *states*, a *transition relation* $T \subseteq S \times S$ such that $\text{id}_S \subseteq T$, a *labelling function* $\text{lab} : (T \setminus \text{id}_S) \rightarrow A$ and a *valuation function* $\text{val} : S \rightarrow |D|^V$.

Unlike in a labelled transition systems of [16], in a component transition systems every two states can be linked by at most one transition. The unlabelled, loop transitions from each state to itself are introduced as “idle” transitions, representing no real action. They allow us to simplify some forthcoming definitions.

As usual, we write $s \longrightarrow s' \in T$ (or just $s \longrightarrow s'$, if T is understood) instead of $\langle s, s' \rangle \in T$. We also write $s \xrightarrow{a} s' \in T$ if $s \longrightarrow s' \in T$ and $\text{lab}(s, s') = a$.

Relations between components and their subcomponents are captured by *component transition system morphisms* (*cts-morphisms* in short). A cts-morphism from \mathbb{T} to \mathbb{T}' is simply a mapping of states of \mathbb{T} to states of \mathbb{T}' that preserves transitions.

Definition 2.2 A *cts-morphism* $f : \mathbb{T} \rightarrow \mathbb{T}'$ between transition systems $\mathbb{T} = \langle A, V, S, T, \text{lab}, \text{val} \rangle$ and $\mathbb{T}' = \langle A', V', S', T', \text{lab}', \text{val}' \rangle$ is a function $f : S \rightarrow S'$ such that, for every $s_1, s_2 \in S$,

$$s_1 \longrightarrow s_2 \in T \text{ implies } f(s_1) \longrightarrow f(s_2)$$

A cts-morphism $f : \mathbb{T} \rightarrow \mathbb{T}'$ induces a mapping of transitions of \mathbb{T} to transitions of \mathbb{T}' . The intuition behind this mapping is that each action of a component may involve a single “sub-action” in its subcomponent. Otherwise, the subcomponent is not engaged in the action and remains idle, which is represented by the component’s transition being mapped to an idle transition of the subcomponent.

Cts-morphisms are thus similar to “classical” transition system morphisms from [16] or functional simulations of Kripke frames from [11]. A notable difference is that a cts-morphism does not relate transition labels, nor the attributes. Relations between elements of component signatures (i.e. actions and attributes) will be easily expressible in the diagram logic.

2.2 Component Models

While a transition system \mathbb{T} models possible state transitions in a component, runs in \mathbb{T} represent *computations* the component may perform.

Formally, a *run* in \mathbb{T} is a finite or infinite sequence $\rho = (s_0 s_1 \dots)$ of states of \mathbb{T} , such that $s_i \longrightarrow s_{i+1}$, for every $i < |\rho| - 1$ (we set $|\rho| = \omega$ for an infinite ρ). For $i < |\rho|$, i -th state of ρ will be denoted by $\rho(i)$ while i -th suffix $(s_i s_{i+1} \dots)$ of ρ will be denoted by ρ^i .

Although we do not impose the usual maximality condition, we still regard runs as representing “complete” computations. A non-maximal run, i.e. one ending in a state from which some transitions go out, represents the possibility that the component ends its execution without taking any of the possible further transitions—

presumably, because they represent actions that require synchronisation with other components that may be not willing to cooperate. It turns out that leaving out the maximality requirement provides an extra expressivity to the logic, as it allows to identify which transitions can be executed by the component independently, and which require interaction with the environment.

Also, the semantics distinguishes “deadlocked” computations looping infinitely in a single state from terminated computations. The former are represented by infinite runs with a constant suffix, while the latter are represented by finite runs.

We consider component models consisting of a component transition system with a distinguished set of runs. This is a customary way of ruling out “infeasible” runs, for instance those representing *unfair* computations (cf. [5]).

Definition 2.3 A *component model* $\mathbb{C} = \langle \mathbb{T}, R \rangle$ consists of a component transition system \mathbb{T} and a suffix-closed set R of runs in \mathbb{T} , such that every state in \mathbb{T} belongs to some run in R .

A *component model morphism* (cm-morphism for short) $f : \langle \mathbb{T}, R \rangle \rightarrow \langle \mathbb{T}', R' \rangle$ is a cts-morphism $f : \mathbb{T} \rightarrow \mathbb{T}'$ such that, for every $\rho \in R$, $f(\rho) \in R'$.

Proposition 2.4 *Component models and their morphisms, with identities and compositions inherited from **Set**, constitute a category, henceforth denoted by **Cmp**.*

*Moreover, the category **Cmp** is complete.*

Limits in **Cmp** can be computed from limits in the underlying category of component transition systems and cts-morphisms, which, in turn, are similar to those in the category of labelled transition systems in [16]. In particular, states of the limit object correspond to tuples of compatible states of diagram components and transitions of the limit object correspond to tuples of compatible transitions of diagram components (see [16] for details).

To obtain a limit in **Cmp**, a limit of underlying transition systems is equipped with the unique set of all those runs, that are mapped to runs in diagram components by projection morphisms. The limit construction in does not determine the actions nor the attributes. Indeed, any two component models differing only in action labels and attribute names and values are isomorphic in **Cmp**.

We omit the full proof of completeness for the lack of space—more details can be found in the extended version of this paper, [17]. Below we only show an example pullback diagram in **Cmp**.

Example 2.5 Let $\mathbb{C} \xrightarrow{f} \mathbb{M} \xleftarrow{g} \mathbb{S}$ be a diagram in **Cmp**, where sets of states and transition relations of \mathbb{C} , \mathbb{S} and \mathbb{M} and cm-morphisms f and g are as shown in Fig. 1 (transitions are drawn with solid arrows and cm-morphisms are drawn with dashed arrows). Idle transitions has been omitted.

Assume that \mathbb{C} has actions *init*, *displ*, *choose* and *load*, \mathbb{S} has actions *invoke*, *proc*, *deliver* and *shutd* and \mathbb{M} has actions *conn*, *term*, *req* and *resp*. For simplicity, the components have no attributes.

The component model \mathbb{C} may represent a graphical web client. After the initialisation (the action *init*) it displays a web page (the action *displ*), allows the user to choose a link to follow (the action *choose*) and loads a new page (the action *load*).

The component model \mathbb{S} may represent a web server. After service invocation

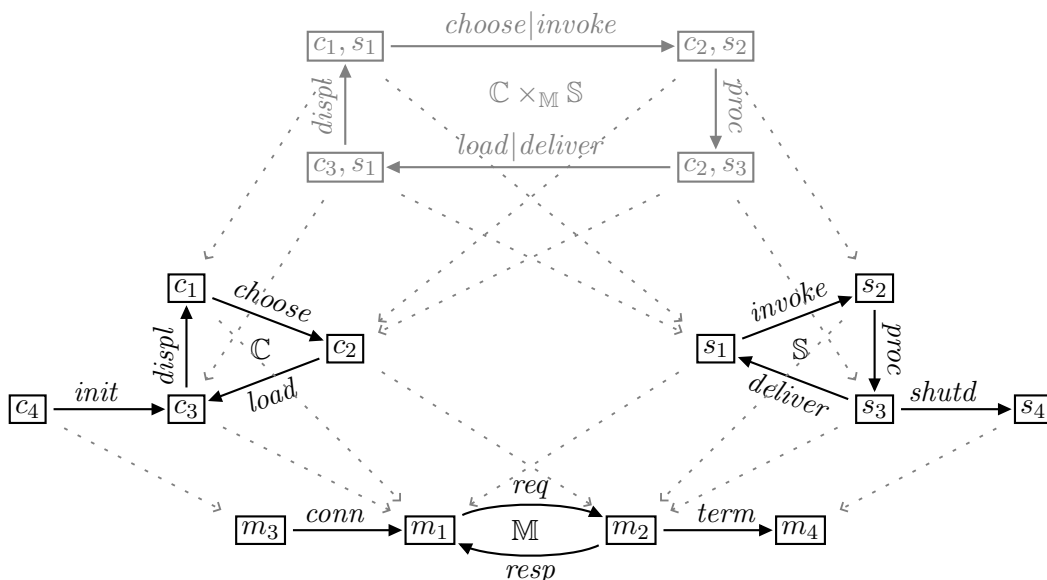


Fig. 1. Example pullback diagram in **Cmp**

(*invoke*) a service request is processed (*proc*) and the response is delivered (*deliver*). The server can be also shut down (*shutd*).

Finally, the component model \mathbb{M} may represent a communication medium, or a channel, between the client and the server. Its actions *req* and *resp* represent the events of transmitting the request data and the response data, respectively. Actions *conn* and *term* represent establishing and terminating the connection between the client and the server.

The limit component model and projection morphisms are also shown on Fig. 1. Its set of actions is not determined, the figure assumes that it contains the names *displ*, *choose|invoke*, *proc* and *load|deliver*.

The limit object represents a system consisting of the client and the server, connected by the communication medium. The action *displ* is “implemented” in the client component, the action *proc*—in the server, whereas actions *choose|invoke* and *load|deliver* arise from pairing of actions in both components and thus involve both the client and the server.

The set of states of the resulting system is obtained as a pullback in **Set** of the state mappings from \mathbb{C} and \mathbb{S} to \mathbb{M} —its states correspond to pairs of states of \mathbb{C} and \mathbb{S} that share a common “sub-state” in \mathbb{M} . Consequently, the states c_4 and s_4 are not represented in the limit component. The limit component can be regarded as representing a *session* involving both the client and the server, and thus it does not contain states in which only one of the components participate.

2.3 Terminal and Initial Components

Particular cases of the limit construction are terminal and initial objects. A terminal component model $\mathbb{C}_\top = \langle \mathbb{T}_\top, R_\top \rangle$ in **Cmp** has a single state t and the set $R_\top = \{t\}^* \cup \{t\}^\omega$ of runs. From any component model \mathbb{C} there is a unique morphism to \mathbb{C}_\top identifying \mathbb{C}_\top as a subcomponent of \mathbb{C} .

An initial component model \mathbb{C}_\perp has no states and thus no runs. It cannot therefore represent any component. It should be emphasised that, in particular, it

does not represent components that loop infinitely in a single state nor the ones that cannot perform any transition, even an idle one—both kinds of components can be represented by component models with a non-empty set of runs.

2.4 Logic for Components

To describe component behaviour we use a variant of the temporal branching-time logic CTL* (cf. e.g. [5]).

Atomic formulae include predicates expressing occurrence of actions and equalities between terms involving component attributes and operations from the underlying data signature. A special atomic formula, \star , distinguishes idle transitions. Compound formulae are built with propositional connectives, the usual temporal operators X (read “next”) and U (read “until”), and the path quantifier A (read “for all runs”).

Additionally, we enrich the language with elements of the first-order logic: variables ranging over data values and the universal quantifier. The fixed domain of quantification is the carrier $|D|$ of the data algebra.

Formally, let A and V be fixed sets of actions and attributes and let \mathcal{OV} be a $srt(\Delta)$ -sorted set of *object variables*. By $\mathcal{T}(V)$ we will denote the $srt(\Delta)$ -sorted set of all terms built from object variables, attributes from V and operation symbols from Δ .

Syntax of *formulae over A and V* , $Frm(A, V)$, is defined by:

$$\phi ::= \perp \mid \phi_1 \rightarrow \phi_2 \mid \forall x^s \phi \mid t_1 = t_2 \mid a \mid \star \mid X\phi \mid \phi_1 U \phi_2 \mid A\phi$$

where $x \in \mathcal{OV}_s$, $t_1, t_2 \in \mathcal{T}(V)$ are terms of the same sort and $a \in A$.

Let $\mathbb{C} = \langle \mathbb{T}, R \rangle$ be a component model with the underlying transition system $\mathbb{T} = \langle A, V, S, T, lab, val \rangle$. Interpretation $\llbracket t \rrbracket_{\xi}^{\mathbb{C}}(s)$ of a term $t \in \mathcal{T}(V)$ in a state $s \in S$ under a valuation $\xi : \mathcal{OV} \rightarrow |D|$ is defined as the standard interpretation of t in the algebra D under the valuation $(\xi + val(s)) : \mathcal{OV} + V \rightarrow |D|$.

Definition 2.6 For a component model \mathbb{C} as above, the satisfaction relation $\models^{\mathbb{C}} \subseteq R \times |D|^{\mathcal{OV}} \times Frm(A, V)$ is defined as follows (obvious cases for propositional connectives are omitted):

$$\begin{aligned} \rho, \xi \models^{\mathbb{C}} \forall x^s \phi & \quad \text{iff } \rho, \xi[d/x] \models^{\mathbb{C}} \phi \text{ for all } d \in |D|_s \\ \rho, \xi \models^{\mathbb{C}} t_1 = t_2 & \quad \text{iff } \llbracket t_1 \rrbracket_{\xi}^{\mathbb{C}}(\rho(0)) = \llbracket t_2 \rrbracket_{\xi}^{\mathbb{C}}(\rho(0)) \\ \rho, \xi \models^{\mathbb{C}} a & \quad \text{iff } |\rho| > 0 \text{ and } lab(\rho(0), \rho(1)) = a \\ \rho, \xi \models^{\mathbb{C}} \star & \quad \text{iff } |\rho| > 0 \text{ and } \rho(0) = \rho(1) \\ \rho, \xi \models^{\mathbb{C}} X\phi & \quad \text{iff } |\rho| > 0 \text{ and } \rho^1, \xi \models^{\mathbb{C}} \phi \\ \rho, \xi \models^{\mathbb{C}} \phi_1 U \phi_2 & \quad \text{iff } \rho^k, \xi \models^{\mathbb{C}} \phi_2 \text{ for some } k \leq |\rho|, \text{ and } \rho^i, \xi \models^{\mathbb{C}} \phi_1 \text{ for all } i < k \\ \rho, \xi \models^{\mathbb{C}} A\phi & \quad \text{iff } \rho', \xi \models^{\mathbb{C}} \phi \text{ for every run } \rho' \in R \text{ such that } \rho'(0) = \rho(0) \end{aligned}$$

Other propositional connectives ($\neg, \rightarrow, \wedge, \vee$) and the existential quantifier are defined as abbreviations.

A formula ϕ is *satisfied in* \mathbb{C} iff $\rho, \xi \models^{\mathbb{C}} \phi$, for every run ρ in R and every valuation $\xi : \mathcal{OV} \rightarrow |D|$.

Example 2.7 In the Example 2.5 we did not fix the sets of runs of the components. Now we can specify the admissible runs using the component logic. Consider the following example properties:

- (1) $display \rightarrow X E choose$
- (2) $load \rightarrow X display$
- (3) $choose \rightarrow X E (\star U load)$
- (4) $choose|invoke \rightarrow X (\star U load)$
- (5) $term \rightarrow X A X \perp$
- (6) $(conn \wedge G X \top) \rightarrow X (req_{\star} \wedge G ((req \rightarrow X resp_{\star}) \wedge (resp \rightarrow X req_{\star})))$

We use the standard abbreviations:

$$E \phi = \neg A \neg \phi, \quad F \phi = \top U \phi, \quad G \phi = \neg F \neg \phi$$

The first three properties concern the component \mathbb{C} (the client). The formula (1) states that immediately after a page is displayed, a link may be chosen by the user. However, as there is no guarantee that the user will want to choose any link, the formula does not rule out runs ending with the action *display*. In contrast, the formula (2) guarantees that after loading, a page will be displayed, independently of the behaviour of other components.

The formula (3) is similar to (1) except that it does not guarantee that *load* can be executed immediately after *choose*.

The next formula can be evaluated against the limit component: while the client itself cannot ensure that *load* will follow *choose* (cf. the formula (3)), when it is paired with a server a guarantee can be given.

The last two formulae describe runs of the component \mathbb{M} . They state that no action (even an idle one) is possible after *term* (the formula (5)) and that every infinite computation consists of the interleaving of the actions *req* and *resp*, possibly preceded by some idle steps. In this last formula, req_{\star} and $resp_{\star}$ abbreviate the formulae $\star U req$ and $\star U resp$, respectively.

2.5 Beyond Limits

As the formula (4) shows, interaction between components can be specified by formulae evaluated in a limit of the corresponding diagram. The next example shows that this is not always the case.

Example 2.8 Interesting properties can also be stated in the context of a diagram depicted schematically in Fig. 2, consisting of a single client component \mathbb{C} and two server components \mathbb{S}_1 and \mathbb{S}_2 , connected by \mathbb{M} . The limit of this diagram is the initial component model, \mathbb{C}_{\perp} , representing no component. In other words, the client component cannot coexist in a single system with both server components, connected via the medium as shown in Fig. 2. However, intuitively, the following statement holds in the diagram:

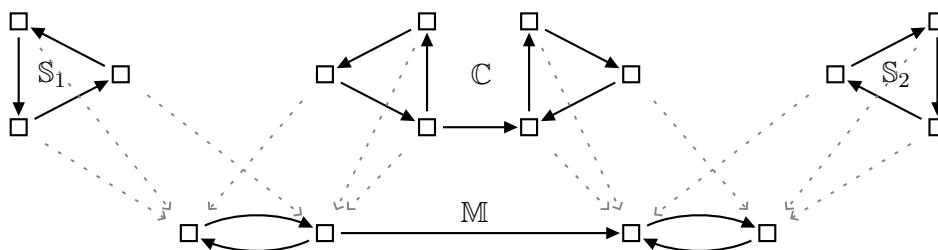


Fig. 2. An example diagram in **Cmp**

The client may first interact with one server and then, after some time, engage in the interaction with the other one.

Moreover, client interaction (a session) with each server can be represented by limit of a subdiagram of the whole diagram, from which the other server is excluded.

The above observations show that limits of diagrams often do not describe component systems in an accurate way. A more expressive formalism can be obtained if we consider whole diagrams as models of component systems.

3 Diagram Models and Diagram Logic

We turn finally to the main topic of this paper—a construction of a “system logic” for describing diagram models.

3.1 Component Diagrams

A preorder category **I** is a category in which, for any two objects c, d , there is at most one morphism from c to d . We write $d \leq_{\mathbf{I}} c$, if such a morphism exists.

A *component diagram* is a functor $\mathbf{C} : \mathbf{I} \rightarrow \mathbf{Cmp}$, where **I** is a small³ preorder category. Objects of **I** can be regarded as *component identities*. For any $i \in |\mathbf{I}|$, the component model

$$\mathbf{C}(i) = \langle A_i, V_i, S_i, T_i, lab_i, val_i \rangle$$

represents the behaviour of the component i . A morphism $k : i \rightarrow j$ in **I** means that the component j is a subcomponent of i . The representation of the relation between i and j is provided by the cm-morphism $\mathbf{C}(k) : \mathbf{C}(i) \rightarrow \mathbf{C}(j)$.

3.2 System Signatures and Frames

A *diagram signature* introduces a number of *roles* for components. Each role comes with its set of actions and attributes.

Definition 3.1 A *diagram signature* $\Sigma = \langle \mathcal{R}, \mathcal{A}, \mathcal{V} \rangle$ consists of

- a set \mathcal{R} (of *roles*),
- a \mathcal{R} -sorted set \mathcal{A} (of *actions*), and

³ A category is *small* if its collection of morphisms is a set.

- a $(\mathcal{R} \times \text{srt}(\Delta))$ -sorted set \mathcal{V} (of *attributes*).

Given a diagram signature Σ , a Σ -*frame* is a component diagram endowed with interpretations of symbols from Σ .

Definition 3.2 A Σ -frame $\mathbb{F} = \langle \mathbf{C}, \{r^{\mathbb{F}} \mid r \in \mathcal{R}\} \rangle$ consists of

- a component diagram $\mathbf{C} : \mathbf{I} \longrightarrow \mathbf{Cmp}$, and
- for any role $r \in \mathcal{R}$, a set $r^{\mathbb{F}} \subseteq |\mathbf{I}|$ such that $\mathcal{A}_r \subseteq A_i$ and $\mathcal{V}_r \subseteq V_i$, for any $i \in r^{\mathbb{F}}$ (recall, that A_i and V_i are actions and attributes of $\mathbf{C}(i)$).

Thus a frame assigns a number of roles (possibly none) to any component $i \in |\mathbf{I}|$. The component model $\mathbf{C}(i)$ has to provide interpretation of actions and attributes from \mathcal{A}_r and \mathcal{V}_r , for any role r assigned to i .

3.3 The Diagram Logic: Syntax

Let $\Sigma = \langle \mathcal{R}, \mathcal{A}, \mathcal{V} \rangle$ be a diagram signature fixed for this section. We assume that \mathcal{CV} is a fixed, \mathcal{R} -sorted set of *component variables*.

To obtain the logic for systems we will modify and extend the component logic. The formulae of the diagram logic will describe “joint runs” of systems built from several components. We will also introduce variables naming individual components, and a suitable binding operator.

Atomic propositions expressing occurrence of actions and component attributes will be qualified with component variable. Thus, instead of simply saying that the action *load* is executed in the system represented by the diagram in Fig. 1, we will say that *load* is executed at c , where the variable c refers to the component \mathbf{C} . Similarly, instead of saying that the attributes p and q have equal values, we will refer to the attribute p of the component c and the attribute q of the component d .

Let \mathcal{QV} denote the set of *qualified attributes* of the form $c.v$, where $c \in \mathcal{CV}_r$ is a component variable and $v \in \mathcal{V}_r$ is an attribute. Terms over Σ , $\mathcal{T}(\Sigma)$, are built from object variables, elements of the set \mathcal{QV} and operations of the data signature Δ .

Temporal operators \mathbf{X} , \mathbf{U} and \mathbf{A} of the component logic are included in the diagram logic. There are also three entirely new constructions: an equality between component variables, $c_1 = c_2$, stating that both variables refer to the same component, an atomic proposition of the form $c_1 \triangleright c_2$, stating that c_2 refers to a subcomponent of c_1 , and the *component quantifier*, written ∇ , binding component variables.

The set of formulae over Σ , $\text{Frm}(\Sigma)$, is defined by the following grammar:

$$\begin{aligned} \phi ::= & \perp \mid \phi_1 \rightarrow \phi_2 \mid \forall x^s \phi \mid t_1 = t_2 \mid c.a \mid c.\star \mid \mathbf{X}\phi_1 \mid \phi_1 \mathbf{U} \phi_2 \mid \mathbf{A}\phi \mid \\ & c_1 = c_2 \mid c_1 \triangleright c_2 \mid \nabla c^r \phi \end{aligned}$$

where $x \in \mathcal{OV}_s$, $t_1, t_2 \in \mathcal{T}(\Sigma)$ are terms of the same sort, $c \in \mathcal{CV}_r$ and $a \in \mathcal{A}_r$, for some $r \in \mathcal{R}$, and $c_1, c_2 \in \mathcal{CV}_{r'}$, for some $r' \in \mathcal{R}$.

Moreover, we require that all component quantifiers in any formula ϕ bind distinct component variables.

3.4 The Diagram Logic: Semantics

Let us fix a Σ -frame \mathbb{F} with an underlying diagram $\mathbf{C} : \mathbf{I} \longrightarrow \mathbf{Cmp}$.

As the example in Fig. 2 shows, the diagram of component models should be not, in general, regarded as a description of a single complex system, with behaviour represented by the diagram's limit. This limit model may turn out to be the initial object \mathbf{C}_\perp , representing no component. Still, one can consider various subdiagrams of \mathbf{C} with non-trivial limits, representing local component configurations.

The key idea is that a formula ϕ with free component variables c_1, \dots, c_n represents a property of a configuration built only from components c_1, \dots, c_n and all their subcomponents. The subdiagram of \mathbf{C} representing the configuration is determined by the *environment*, i.e. a partial function $\eta : \mathcal{CV} \rightarrow |\mathbf{I}|$ that maps component variables to objects in \mathbf{I} . The diagram $\mathbf{C}|_\eta$ is obtained by restricting \mathbf{C} to the subcategory of \mathbf{I} with the set of objects

$$\{i \in |\mathbf{I}| \mid i \leq_{\mathbf{I}} \eta(c), \text{ for some } c \in \text{dom}(\eta)\}$$

In other words, $\mathbf{C}|_\eta$ is the smallest subdiagram of \mathbf{C} containing components named with variables bound by η and all their subcomponents.

The behaviour of the configuration determined by η is represented by the limit model of $\mathbf{C}|_\eta$, denoted by $L_{\mathbf{C}}(\eta) = \langle \langle A_\eta, V_\eta, S_\eta, T_\eta, \text{lab}_\eta, \text{val}_\eta \rangle, R_\eta \rangle$. For any $c \in \text{dom}(\eta)$ there is a projection morphism $\lambda_c^\eta : L_{\mathbf{C}}(\eta) \rightarrow \mathbf{C}(\eta(c))$, i.e., a state mapping $\lambda_c^\eta : S_\eta \rightarrow S_{\eta(c)}$.

The formula ϕ can be then evaluated against a run in $L_{\mathbf{C}}(\eta)$, representing a joint component execution. In Section 2.2 we noted that every run in a limit model is mapped to runs in diagram components by projection morphisms.

In order to define the semantics of the component quantifier we need to define the notion of *extension* of a run. Let η_i^c be the environment extending η with a binding of a component variable $c \notin \text{dom}(\eta)$ to $i \in |\mathbf{I}|$. Since $\mathbf{C}|_\eta$ is a subdiagram of $\mathbf{C}|_{\eta_i^c}$, there exists a unique mediating morphism $\mu_{c,i}^\eta : L_{\mathbf{C}}(\eta_i^c) \rightarrow L_{\mathbf{C}}(\eta)$ between the limit models. A run ρ' in $L_{\mathbf{C}}(\eta_i^c)$ is a *i-extension* of a run ρ in $L_{\mathbf{C}}(\eta)$ if $\mu_{c,i}^\eta(\rho') = \rho$.

Definition 3.3 The satisfaction relation $\models^{\mathbb{F}}$ between an environment η , a run ρ in $L_{\mathbf{C}}(\eta)$, a valuation ξ and a Σ -formula ϕ is defined as in Figure 3 (cases for propositional connectives and the first-order quantifier are omitted).

The clauses for temporal operators \mathbf{X} , \mathbf{U} and \mathbf{A} are as in the definition of satisfaction relation for component logic.

Interpreting term equalities involve straightforward term evaluation: value of $c.v$ in a state s of the limit model is the value of the attribute v in s projected to state $\lambda_c^\eta(s)$ of $\mathbf{C}(\eta(c))$. Similarly, $c.a$ holds for the joint run if its first transition is mapped to a transition in $\mathbf{C}(\eta(c))$ labelled with a .

The interpretation of $c_1 = c_2$ and $c_1 \triangleright c_2$ does not depend on the run.

Finally, $\nabla c^r \phi$ holds for a joint run ρ if ϕ holds for all extensions of ρ to all larger systems obtained by adding some new component c of the role r .

$$\begin{array}{ll}
 \eta, \rho, \xi \models^{\mathbb{F}} \mathbf{X} \phi & \text{iff } |\rho| > 0 \text{ and } \eta, \rho^1, \xi \models^{\mathbb{F}} \phi \\
 \eta, \rho, \xi \models^{\mathbb{F}} \phi_1 \mathbf{U} \phi_2 & \text{iff } \eta, \rho^k, \xi \models^{\mathbb{F}} \phi_2 \text{ for some } k \leq |\rho|, \text{ and } \eta, \rho^i, \xi \models^{\mathbb{F}} \phi_1 \text{ for all } i < k \\
 \eta, \rho, \xi \models^{\mathbb{F}} \mathbf{A} \phi & \text{iff } \eta, \rho', \xi \models^{\mathbb{F}} \phi \text{ for every run } \rho' \text{ in } L_{\mathbf{C}}(\eta) \text{ such that } \rho'(0) = \rho(0) \\
 \eta, \rho, \xi \models^{\mathbb{F}} t_1 = t_2 & \text{iff } \llbracket t_1 \rrbracket_{\xi}^{\eta}(\rho(0)) = \llbracket t_2 \rrbracket_{\xi}^{\eta}(\rho(0)) \\
 \eta, \rho, \xi \models^{\mathbb{F}} c.a & \text{iff } |\rho| > 0 \text{ and } \lambda_c^{\eta}(\rho(0)) \xrightarrow{a} \lambda_c^{\eta}(\rho(1)) \in T_{\eta(c)} \\
 \eta, \rho, \xi \models^{\mathbb{F}} c.\star & \text{iff } |\rho| > 0 \text{ and } \lambda_c^{\eta}(\rho(0)) = \lambda_c^{\eta}(\rho(1)) \\
 \eta, \rho, \xi \models^{\mathbb{F}} c_1 = c_2 & \text{iff } \eta(c_1) = \eta(c_2) \\
 \eta, \rho, \xi \models^{\mathbb{F}} c_1 \triangleright c_2 & \text{iff } \eta(c_2) \leq_{\mathbf{I}} \eta(c_1) \\
 \eta, \rho, \xi \models^{\mathbb{F}} \nabla c^r \phi & \text{iff } \eta_i^c, \rho', \xi \models^{\mathbb{F}} \phi \text{ for all } i \in r^{\mathbb{F}} \text{ and every } i\text{-extension } \rho' \text{ of } \rho \text{ in } L_{\mathbf{C}}(\eta_i^c)
 \end{array}$$

Fig. 3. Satisfaction relation for the diagram logic

3.5 Sentences and Satisfaction in a Frame

As usual, a *sentence* over Σ is a formula with no free variables (of any kind). A sentence ϕ represents a property of the subdiagram determined by the empty environment. This is the empty subdiagram and its limit is $\mathbf{C}_{\top} = \langle \mathbb{T}_{\top}, R_{\top} \rangle$, the terminal object in **Cmp**.

Consequently, a sentence ϕ is *satisfied in the frame* \mathbb{F} if $\emptyset, \rho, \xi \models^{\mathbb{F}} \phi$ for every run $\rho \in R_{\top}$ and for every valuation ξ (obviously, interpretation of a sentence does not depend on ξ).

4 Example: Clients and Servers

To illustrate the expressive power of the diagram logic we describe a system consisting of *client* and *server* components that communicate with each other by means of end-to-end *sessions*.

In this example, a system signature consists of the roles *Clnt* (for clients), *Srvr* (for server) and *Sess* (for sessions). Components with the role *Srvr* have the attribute *addr* of sort $\text{Addr} \in \text{srt}(\Delta)$.

Some example properties, written as sentences of the diagram logic

- (1) *Every server has a unique address.*

$$\nabla s_1^{Srvr} \nabla s_2^{Srvr} (s_1.addr = s_2.addr \rightarrow s_1 = s_2)$$

- (2) *The server's address does not change in time.*

$$\nabla s^{Srvr} \forall a^{Addr} (s.addr = a \rightarrow \mathbf{G} s.addr = a)$$

An interesting point here is that a frame may include two *Srvr* components with the same value of the *addr* attribute, and still satisfy (1). To see this, assume that the frame in question has the underlying diagram as shown in Fig. 2, and that

components \mathbb{S}_1 and \mathbb{S}_2 of the role $Srvr$ have the same $addr$ attribute. The sentence (1) will be satisfied in this frame iff the formula $s_1.addr = s_2.addr \rightarrow s_1 = s_2$ is satisfied in every environment η binding variables s_1 and s_2 . The interesting case is when $\eta(s_1) \neq \eta(s_2)$. The limit of the subdiagram containing $\eta(s_1)$ and $\eta(s_2)$ is then the empty initial model with no runs. All its runs extend the runs of the terminal model, and hence the whole sentence (1) is satisfied.

This phenomenon can be explained by observing that \mathbb{S}_1 and \mathbb{S}_2 represent components that cannot coexist in one system—they belong to two different “alternative worlds”.

A properties illustrating the use of the \triangleright operator are:

- (3) *Every session contains exactly one client.*

$$\nabla e^{Sess} \Delta c_1^{Clnt} (e \triangleright c_1 \wedge \nabla c_2^{Clnt} e \triangleright c_2 \rightarrow c_1 = c_2)$$

(A formula $\Delta c^r \phi$ abbreviates $\neg \nabla c^r \neg \phi$.) Similarly, it can be stated that every session has exactly one server.

- (4) *A client can participate in one session at a time.*

$$\nabla c^{Clnt} \nabla e_1^{Sess} \nabla e_2^{Sess} e_1 \triangleright c \wedge e_2 \triangleright c \rightarrow e_1 = e_2$$

Again, a frame satisfying (4) may contain several $Sess$ components sharing a single $Clnt$ subcomponent—just as \mathbb{S}_1 and \mathbb{S}_2 in Fig. 2 share M . What is required is that no run involving a $Clnt$ component may be extended to a run involving two distinct $Sess$ components.

The meaning of (3) is more straightforward: in a frame satisfying this sentence, every $Sess$ component must have exactly one $Clnt$ subcomponent.

Further, assume that the role $Clnt$ has an attribute srv_addr of sort $Addr$ and an action $conn$.

- (5) *The action $conn$ can only be executed by a component that does not participate in a session.*

$$\nabla c^{Clnt} (c.conn \rightarrow \nabla e^{Sess} \neg e \triangleright c)$$

- (6) *After executing $conn$, the client establishes a session with a server with the address stored in the attribute srv_addr prior to the execution of $conn$.*

$$\begin{aligned} \nabla c^{Clnt} \forall a^{Addr} (c.conn \wedge c.srv_addr = a \rightarrow \\ \times \Delta e^{Sess} \Delta s^{Srvr} (e \triangleright c \wedge e \triangleright s \wedge s.addr = a)) \end{aligned}$$

In order to describe interaction within session, we add actions $send$ and $recv$ to the roles $Clnt$ and $Srvr$, and actions req , $resp$ to the role $Sess$.

- (7) *Actions req and $resp$ of sessions involve actions $send$ and $recv$ in session sub-components.*

$$\begin{aligned} \nabla e^{Sess} \nabla c^{Clnt} \nabla s^{Srvr} (e \triangleright c \wedge e \triangleright s \rightarrow \\ (s.req \rightarrow c.send \wedge s.recv) \wedge (s.resp \rightarrow s.send \wedge c.recv)) \end{aligned}$$

The next property will be more problematic. We assume each session has an identifier, represented by the attribute $sess_id$ of role $Sess$ with the sort $Id \in srt(\Delta)$. In practise, session identifiers may be used to determine to which session a low-level network message belongs. One may then require that any two sessions, served by the same server, have different identifiers, even if they do not overlap in time (i.e., one terminates before the other starts). It is not enough to write

$$\nabla e_1^{Sess} \nabla e_2^{Sess} (e_1.sess_id = e_2.sess_id \wedge (\Delta s^{Srvr} e_1 \triangleright s \wedge e_2 \triangleright s) \rightarrow e_1 = e_2)$$

since that would guarantee uniqueness of identifiers for sessions that exist at the same moment. One should also add

$$\begin{aligned} \nabla s^{Srvr} \forall i^{Id} (\Delta e_1^{Sess} (e_1 \triangleright s \wedge e_1.sess_id = i \wedge \mathbf{X} \perp)) \\ \rightarrow \mathbf{X} \mathbf{G} \nabla e_2^{Sess} (e_2 \triangleright s \rightarrow e_2.sess_id \neq i) \end{aligned}$$

The antecedent of the above implication is satisfied by runs starting in a terminal state of a session with an identifier i . Any sessions that exists afterwards will have an identifier different from i .

In the final example we assume that the signature contains action symbols of components in Figure 1, with appropriate roles.

- (8) *The client itself cannot guarantee that the action load will be executed. However, load will be executed if the client is involved in a session.*

$$\begin{aligned} \nabla c^{Clnt} \mathbf{E} \mathbf{X} c.load \rightarrow \mathbf{E} \mathbf{X} \perp \\ \nabla s^{Sess} \nabla c^{Clnt} s \triangleright c \wedge \mathbf{E} \mathbf{X} c.load \rightarrow \mathbf{X} c.load \end{aligned}$$

5 Expressivity of Diagram Logic

In this section we summarise the results concerning the expressivity of the diagram logic. The main result is that the diagram logic is incomplete, that is, the set of valid sentences is not recursively enumerable.

The diagram logic has limited expressive power as far as the shapes of diagrams are concerned. For instance, as illustrated by the formula (1) in Section 4, a diagram with a single component of the role $Sess$ may be logically equivalent to a diagram with multiple $Sess$ components, as long as only one of them may belong to a configuration.

This observation can be formalised by defining a suitable notion of bisimilarity relating logically equivalent diagrams. Details can be found in [17], here we only report one of the consequences.

Proposition 5.1 *Let Σ be a system signature and let Φ be a set of sentences over Σ . If Φ is satisfied by a frame \mathbb{F} with the underlying diagram of cardinality α then Φ is satisfied by some frame with the underlying diagram of cardinality $\alpha \cdot \beta$, for every cardinal $\beta > 0$.*

It is well known that first-order temporal logics (with the “until” or “finally”) are incomplete (cf. e.g. [14]). The diagram logic inherits this property.

Proposition 5.2 *There exist a data signature Δ and a system signature Σ such that the set of sentences satisfied in every frame over Σ is not recursively enumerable.*

The proof relies on the fact that the standard model of arithmetic can be characterised in such logics up to isomorphism ([14]).

What is more interesting is that the proposition applies also to a fragment of the diagram logic not including first-order constructs, i.e. with no object variables and quantifiers. It turns out that a first-order specification can be “encoded” in this fragment of the diagram logic (details can be found in the extended version of this paper, [17]).

6 Discussion and Related Work

The contribution of this paper is the logic for the specification of interacting component systems. In contrast with the traditional categorical approach, where temporal formulae are evaluated in a limit of the diagram representing the system, the diagram logic allows one to describe the diagram itself. In our logic, a single model (i.e. a diagram) may represent many alternative configurations of components—there may be no single “universal” configuration, containing all others and providing a global point of view. A component may evolve “in two dimensions”: as usual in modal and temporal logics, it may have many alternative futures; additionally, it may belong to several alternative, mutually exclusive configurations. We are not aware of any other formalism with this latter feature.

Such a model seems to represent in a natural way a decentralised community of components, where various possible configurations may emerge and there is no need of a single “global” view.

Some categorical formalisms for the specification of component systems has been mentioned in Section 1 ([7], [6],[13],[15]). There is also the so called “state as algebra” approach, in which a system is represented by a transition system with states labelled by algebras or first-order structures ([1]). While such algebraic formalisms allow one to describe structure of the system using the full first-order logic, they are not as compositional as categorical models. They also impose the global, monolithic view of system’s state.

Structural and behavioural aspects can also be interleaved in *spatial logics* ([3,4]). However, spatial logics are closely related to particular process calculi, e.g. *mobile ambients* ([4]) or versions of *π -calculus* ([3])—the syntax of those logics follows closely the syntax of processes while their semantics is induced by operational semantics of the underlying calculi. Usually, spatial logics do not allow one to quantify over components or processes.

To some extent, system structure and behaviour can be described in the logics described in [13] and [10]. However, those logics are restricted to linear-time semantics. Additionally, the second one presupposes a tree-like, hierarchical system structure.

One important result we have obtained that did not fit in the framework of this paper is the formalisation of the diagram logic as an *institution* ([2]), with frames

used as models. This opens the possibility of using various institution-independent specification techniques and tools ([12]) to build both system specifications and models in an incremental, structured way.

In spite of the incompleteness result, we may still hope to develop useful inference rules and proof techniques for the logic. It may be possible to find a complete proof system for some useful fragments of the diagram logic—a logic with no U (until) is one candidate.

References

- [1] Astesiano, E. and E. Zucca, *D-oids: a model for dynamic data-types*, *Mathematical Structures in Computer Science* **5** (1995), pp. 257–282.
- [2] Burstall, R. and J. Goguen., *Institutions: Abstract model theory for specification and programming.*, *Journal of the ACM* **39** (1992), pp. 95–146.
- [3] Caires, L. and L. Cardelli, *A spatial logic for concurrency (part i)*, *Information and Copmputation* **186** (2003), pp. 194–235.
- [4] Cardelli, L. and A. Gordon, *Anytime, anywhere: Modal logics for mobile ambients*, in: M. Wegman and T. Reps, editors, *POPL'00: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (2000), pp. 365–377.
- [5] Emerson, E. A., *Temporal and modal logic*, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematic*, Elsevier/The MIT Press, 1990 pp. 995–1072.
- [6] Fiadeiro, J., A. Lopes and M. Wermelinger, *A mathematical semantics for architectural connectors.*, in: R. Backhouse and J. Gibbons, editors, *Generic Programming*, *Lecture Notes in Computer Science* **2793** (2003), pp. 178–221.
- [7] Fiadeiro, J. and T. Maibaum, *Temporal theories as modularisation units for concurrent system specification*, *Formal Aspects of Computing* **4** (1992), pp. 239–272.
- [8] Goguen, J., *Sheaf semantics for concurrent interacting objects*, *Mathematical Structures in Computer Science* **2** (1992), pp. 159–191.
- [9] Goguen, J. and S. Ginali, *A categorical approach to general systems theory*, in: G. Klir, editor, *Applied General Systems Research*, Plenum, 1978 pp. 257–270.
- [10] Merz, S., M. Wirsing and J. Zappe, *A spatio-temporal logic for the specification and refinement of mobile systems*, in: *FASE 2003*, *LNCS* **2621** (2003), pp. 87–101.
- [11] Palomino, M., J. Meseguer and N. Martí-Oliet, *A categorical approach to simulations*, in: J. Fiadeiro, N. Harman, M. Roggenbach and J. Rutten, editors, *Algebra and Coalgebra in Computer Science: First International Conference, CALCO 2005, Swansea, UK, September 3-6, 2005, Proceedings.*, *LNCS* **3629** (2005), pp. 313–330.
- [12] Sannella, D. and A. Tarlecki, *Specifications in an arbitrary institution*, *Information and Computation* **76** (1988), pp. 165–210.
- [13] Sernadas, A., C. Sernadas and J. F. Costa, *Object specification logic*, *Journal of Logic and Computation* **5** (1995), pp. 603–630.
- [14] Szalas, A. and L. Holenderski, *Incompleteness of first-order temporal logic with until.*, *Theor. Comput. Sci.* **57** (1988), pp. 317–325.
- [15] Vickers, S. and G. Hill, *Presheaves as configured specifications*, *Formal Aspects of Computing* **13** (2001), pp. 32–49.
- [16] Winskel, G. and M. Nielsen, *Models for concurrency*, in: S. Abramsky, D. Gabbay and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, Oxford University Press, 1995 pp. 1–148.
- [17] Zawłocki, A., *Diagram models for interacting components: Extended version* (2006), unpublished. URL www.mimuw.edu.pl/~zawlocki/papers/facs06-extended.pdf