A Semantic Model for Computer-Based Spatio-Temporal Systems

Benjamin Hummel
Technische Universität München
Institut für Informatik
Garching bei München, Germany
hummelb@in.tum.de

Abstract

Over the last decades lots of techniques have been developed for modeling, analyzing, and verifying software. For embedded or computer-based systems, however, the software is only one part of the entire system, which often has little or no meaning when examined in isolation without considering the remaining parts of the system. This makes it hard, if not impossible, to judge the correctness of software without a thorough understanding of its environment.

A natural solution to this problem is to not only capture the software part but the entire system by suitable design models. However, to be useful such a model has to be supported by semantics which unambiguously define its meaning. In this paper we present such a semantic model which captures temporal and spatial aspects of a system, which are important if the system deals with the manipulation of rigid objects, as typically found in the domain of industrial automation.

1. Introduction

When dealing with embedded or computer-based systems, usually it is not sufficient to treat the software part alone, but rather the entire system has to be considered. This includes mechanical and electronic parts which are used to perceive and manipulate its surroundings, but also assumptions about the environment, such as the presence of gravity or a certain amount of air pressure. Such an integrated view is already needed in the requirements phase, where requirements are usually not formulated with respect to the software but with respect to the complete system. Often it is not even possible to break requirements down to software requirements early on, as it might not be clear before the design phase which parts of the system's functionality are performed by some clever interaction of mechanics and electronics or rather realized mostly in software. The problem does not become easier, when it goes down to testing or formal verification, as the output signals generated by controller software are hardly interpretable without a deep understanding of the complete system.

To distribute information about the system between engineers of the different disciplines involved, including software and systems engineers, we argue that it should be captured by semantically founded models. We emphasize the semantic foundation here, as it ensures that everyone has the same interpretation of a model and allows the application of advanced analysis and verification techniques, including the generation of test-cases or property proofs by model checking. Existing modeling techniques however either fail to capture the behavioral aspects of the system, which are usually the most important from a software engineer's perspective, or lack a formal semantic foundation. The first class includes many of the CAD models used in mechanical and electrical engineering, while the second class includes well-known industrial languages, such as SysML [17].

As the aspects of a system which should be captured in a design model are numerous, it is complicated to build a modeling technique which can express all off them. So as a first step we concentrate on spatio-temporal systems, where we describe spatial properties of a system over time (and of course computation and communication). For many interesting domains such models are sufficient to describe the behavior of the entire system, as other physical quantities, such as temperature or electromagnetic effects, do not significantly influence these systems or can be neglected under normal operating conditions. An example for this is industrial automation, where the purpose of a system is mostly to transport, manipulate, and assemble certain products. This is performed mostly by spatial effects, *i.e.*, by pushing other objects or intersecting the ray of a photoelectric barrier.

Contribution This paper introduces a semantical basis for modeling computer-based spatio-temporal systems by extending the ideas of stream-based interface descriptions as introduced in [4]. The semantic model captures the positions of objects over time and supports basic concepts re-

quired for the modeling of realistic systems, such as the collision between solid objects, the detection of objects in certain locations, or modeling of dependent motions (motion hierarchies). At the same time the usual primitives needed to express computation and communication are available. As our focus is on space, we use a simple discrete model of time, but support both discrete and continuous models of space. We describe a set of operations, which can be used to systematically build spatio-temporal systems from smaller parts and thus support the system decomposition and reuse on the model level. Using our model we formalize relevant properties of spatio-temporal systems.

Outline The next section discusses work related to the topic of this paper. In Sec. 3 we introduce a formalization of space to build upon, followed by the presentation of the semantic model in Sec. 4. To clarify the concepts, an example is provided in Sec. 5. Sec. 6 discusses and formalizes properties of spatio-temporal systems, before we finally give a perspective to future work and conclude in Secs. 7 and 8.

2. Related work

The number of modeling approaches for embedded software and computer-based systems proposed over the last decades is overwhelming. Thus we concentrate on those techniques, we consider the most prominent in the field or most relevant to our work. Additionally, techniques without clear semantics, such as SysML [17], are not considered here.

The most general models are offered by the area of hybrid systems, which combine discrete and continuous aspects. Most work focuses on different flavors of hybrid automata [11, 16], although there are also attempts of using hybrid process algebras [5] for reasoning about systems. While those models have shown their strength when describing control theoretic problems or including continuous time in discrete systems, their application to spatio-temporal systems is not straight forward, as the notion of space allocation, collision, or complex transformations have to be mapped to the rather primitive (although expressive) means available in these models.

For the explicit treatment of space both algebraic and logic approaches have been proposed [6, 15] and in [8] spatial logic is combined with temporal logic to form a *spatio-temporal logic*. However, those papers either concentrate on capturing spatial knowledge and reasoning about it or explore complexity and decidability of these logics. Our focus is more on the specification of system behavior, *i.e.*, instead of capturing *incomplete* knowledge we intend to describe all possible reactions of a system. Additionally, the cited work does not include computation and communication, as is needed when describing computer-based systems. Nonetheless, these techniques can form a basis for the analysis of spatio-temporal systems.

In the area of mechatronic systems modeling the systems are either described at an abstraction level that completely neglects spatial aspects [13, 19] or by using physical simulation systems, such as Modelica [20, 7], where even physical effects like friction or mass inertia are included. For design models we consider the intermediate approach most appropriate, as for many systems spatial effects (collision, activation of sensors) are vital for performing correctly, but on the other hand the effort needed to build a physical simulation is too much in the design stage.

Finally, all techniques used for modeling (embedded) software systems, such as Statecharts [10], the synchronous LUSTRE [9], or the asynchronous FOCUS [4], can to some extent be used for modeling spatio-temporal systems by encoding spatial properties by suitable data structures and treating them as data state. However, this encoding is rather tedious and obstructs the view to the underlying problem. By extending these formalisms, the support for spatio-temporal systems can be greatly improved. Actually the work presented here can be seen as a spatio-temporal extension of FOCUS.

3. Formalizing space

When reasoning about space, we first need a formal model of space. We model space by a tuple $(V, \bowtie, \sqcup, \mathcal{T})$, where

- V is a set of *volumes* with zero element $0_V \in V$,
- $\bowtie \subseteq V \times V$ is called *collision relation*,
- $\sqcup : V \times V \to V$ is the *volume union operator*,
- $\mathcal{T} \subseteq \{V \to V\}$ defines valid volume transformations.

We require the collision relation to be reflexive for non-empty volumes, the empty volume to collide with nothing, and symmetric, *i.e.*, for all $u, v \in V$

$$v \neq 0_V \;\Leftrightarrow\; v \bowtie v \;\;,$$

$$\neg(0_V \bowtie v \;\vee\; v \bowtie 0_V) \;\;,$$

$$u \bowtie v \;\Leftrightarrow\; v \bowtie u \;\;.$$

Additionally $(V, \sqcup)$ has to be a commutative monoid with identity element 0_V and $\sqcup$ is required to be idempotent. Finally the following law of distributivity has to hold for all $u, v, w \in V$:

$$u \bowtie v \;\vee\; u \bowtie w \quad \Leftrightarrow \quad u \bowtie (v \sqcup w)$$

As a special case this implies that $v \bowtie (v \sqcup u)$ holds, *i.e.*, adding more space to an object will not make it smaller.

For the transformation functions in $\mathcal{T}$ we require them to be closed under composition (denoted by $\circ$[1]), *i.e.*, $\{t_1 \circ t_2 \mid t_1, t_2 \in \mathcal{T}\} \subseteq \mathcal{T}$. Additionally transformations may not distort volumes, which means that for all $T \in \mathcal{T}$ and $u, v \in V$

$$\begin{aligned} u \bowtie v &\Leftrightarrow T(u) \bowtie T(v) \ , \\ T(u \sqcup v) &= T(u) \sqcup T(v) \ . \end{aligned}$$

Intuitively each function in $\mathcal{T}$ describes a transformation of one or more volumes, which will later be used, *e.g.*, to encode the current amount of rotation of a machine part.

Using these basic axioms we can introduce the *over-approximation relation* $\sqsubseteq \subseteq V \times V$ defined by

$$u \sqsubseteq v \quad :\Leftrightarrow \quad \forall w \in V : \ w \bowtie u \Rightarrow w \bowtie v \ ,$$

which states that v over-approximates u, if each object colliding with u also collides with v. So spatial over-approximation can be interpreted as a kind of super set relation and forms a partial order on V.

An example for a space fulfilling all of these requirements is $(V, \bowtie, \sqcup, \mathcal{T})$, where V is the set of compact subsets of $\mathbb{R}^3$, $0_V = \emptyset$, $\bowtie := \{(v_1, v_2) \in V \times V \mid v_1 \cap v_2 \neq \emptyset\}$, $\sqcup(v_1, v_2) := v_1 \cup v_2$, and $\mathcal{T}$ the set of translations and rotations in $\mathbb{R}^3$. This example captures the intuition about Euclidean space with rigid objects (*i.e.*, a transformation does not change the size of an object). Other examples where $\sqcup$ and $\bowtie$ are not reduced to set operations can be found in the area of *constructive solid geometry (CSG)* [12].

We always describe spatio-temporal systems with respect to a single space, which has to be declared beforehand. For the remainder of this paper we assume a space $(V, \bowtie, \sqcup, \mathcal{T})$ to be fixed.

4. Modeling spatio-temporal components

This section introduces the semantic framework we are using for modeling spatio-temporal systems. The entire system is itself treated as a so called *spatio-temporal component* which may either be described directly (atomic component) or by composition of other components. This allows the decomposition of larger systems to manage their complexity, but also supports the creation of a library of basic building blocks which are used and reused over different models.

We will start our presentation by recapitulating the foundations for our model followed by an introduction to the notion of an interface for spatio-temporal components. The remaining subsections then describe operators which can be used to construct new components from other components.

[1] We use the definition of $(g \circ f)(x) := g(f(x))$ here; some textbooks use the reverse definition.

4.1. Preliminaries

Before presenting our model, we introduce some concepts and notation. For a set Σ we denote its power set by $\mathcal{P}(\Sigma)$ and the set of infinite sequences over Σ by $\Sigma^\infty := \{\mathbb{N} \to \Sigma\}$, *i.e.*, they are identified with mappings $\mathbb{N} \to \Sigma$. We call those infinite sequences *streams*. For a stream s we denote its prefix of length t by $s \downarrow t$ and instead of $s(t)$ we write $s.t$ most of the time. We use $f|_D$ to denote the restriction of a function f to the new domain D.

We assume a set TYPES of type symbols and universe DATA of all possible data values. Here we are only interested in assigning the set of valid data elements to a type (its *carrier set*) by means of a function car : TYPES $\to \mathcal{P}(\text{DATA})$. More elaborate type systems might require a set of operations for each type and relationships between types.

We call a set L of channel labels a *typed channel set*, if there is a mapping $\text{type}_L : L \to \text{TYPES}$, which assigns to each channel its type. For a typed channel set L, a *channel valuation* is a type correct mapping from channels to data streams, so the set of all channel valuations of L, denoted by $\vec{L}$, is given by

$$\{x : L \to \text{DATA}^\infty \mid \forall l \in L : \ x(l) \in \text{car}(\text{type}_L(l))^\infty\} \ .$$

With these concepts a component in [4] is described by two typed channel sets I and O (its syntactic interface describing input and output channels) and a relation between $\vec{I}$ and $\vec{O}$ (its semantic interface). The reason to use streams for describing the semantic interface is to capture the flow of time, *i.e.*, the streams are interpreted as input and output values over time.

4.2. Component interfaces

Spatio-temporal components are described by their syntactic and semantic interface. For atomic components this means that the function introduced below has to be described by means of mathematical notations or some other suitable description technique, such as some adapted automaton model. For operators on components this indicates that the resulting component has to be expressible in terms of this interface description.

We model the interface of spatio-temporal components by their *data interface* (*i.e.*, input and output ports), the space they allocate (called *parts*), *detectors* (volumes detecting the presence of other parts, which is modeled by a collision between the volumes of the part and the detector), and *movers* which may affect the position of other components, *i.e.*, other components can be connected to these movers and their position is then affected by the component and the state of its mover. For mechatronic systems, detection areas correspond to the working area of sensors (*e.g.*, the light ray of a photoelectric barrier or the area covered by

a proximity sensor), parts model the rigid elements (such as steel frames or the housing of sensors), and movers represent actuators, such as pneumatic cylinders or motors with gearboxes.

Following [4] we distinguish the syntactical interface, which describes the static aspects of a component (which ports and parts are present) and the semantic or behavioral interface (what reactions are expected for given stimuli). The interface description of a component contains all details required to use it and analyze its behavior, but does not disclose any details on how this behavior is achieved (implementation).

The syntactic interface of a spatio-temporal component C is described by a tuple (I, O, D, P, M), with typed channel sets I and O, and sets of detectors D, parts P, and movers M. All of these sets can be interpreted as labels used to identify the respective items.

The semantics of C are defined by a function

$$C : \vec{I} \times \mathcal{A}(D) \to \mathcal{P}(\vec{O} \times \vec{D} \times \vec{P} \times \vec{M} \times V^{\infty}) \ ,$$

where

- $\vec{I}$ and $\vec{O}$ are channel valuations as introduced before,
- $\mathcal{A}(D) := \{D \to \mathbb{B}^{\infty}\}$ are detector activations[2],
- $\vec{D} := \{D \to V^{\infty}\}$ are detector positions/volumes,
- $\vec{P} := \{P \to V^{\infty}\}$ are part positions/volumes,
- $\vec{M} := \{M \to \mathcal{T}^{\infty}\}$ are transformations exercised by movers, and
- the final V^{∞} marks the so called "forbidden" area, *i.e.*, the volume that is expected to stay empty for the component to work correctly. The idea is to explicity model the space which a system expects to be free of external impact to actually deliver the promised service.

Intuitively a spatio-temporal component receives data input and reads information via its detectors ($\mathcal{A}(D)$). As a response it outputs data, changes the positions of its detectors and rigid parts, modifies the transformations caused by its movers, and adjusts its notion about a region in space which is expected to be kept free of external objects for the component to work correctly. As we are interested in the behavior of the system over a possibly infinite time, all inputs and outputs are not single elements but streams (or sets of streams)[3]. These are interpreted under the assumption of a discrete time, *i.e.*, all stream elements are associated with a certain time slot. This assumption is valid if the time steps are small enough, however more complicated timing models and their relationships are discussed in [1] and seem to carry over to our model if needed. For the output the power set is used to both model partiality ($C(i, a) = \emptyset$), as not all inputs may lead to a valid configuration of the system, and non-determinism ($|C(i, a)| > 1$) as a result of underspecification or abstraction of physical phenomena.

Furthermore we require the component to be strongly causal for valid inputs (*i.e.*, those with non-empty output sets). We adapt the definition from [3] to our case, which means for inputs $(i_1, a_1), (i_2, a_2) \in \vec{I} \times \mathcal{A}(D)$ with $C(i_1, a_1) \neq \emptyset$ and $C(i_2, a_2) \neq \emptyset$ and any $t \in \mathbb{N}$ we require

$$\begin{aligned}&(i_1, a_1)\downarrow t = (i_2, a_2)\downarrow t \ \Rightarrow \\ &\ \{y\downarrow(t{+}1) \mid y{\in}C(i_1, a_1)\} = \{y\downarrow(t{+}1) \mid y{\in}C(i_2, a_2)\},\end{aligned}$$

i.e., the outputs and positions at some time $t + 1$ may only depend on previous inputs and detector activations (on or before time t) but not on simultaneous or future input. This is crucial to ensure the soundness of the composition of components and can be interpreted as the minimal reaction delay of a system.

The following sections describe operators which can be used to construct spatio-temporal components from other spatio-temporal components and thus support the decomposition of a complex system into more manageable parts.

4.3. Parallel composition

The parallel composition of spatio-temporal components corresponds to the case of putting both components next to each other. So in terms of data exchange nothing interesting happens, but the volumes of the parts may activate the detectors of the other component. Additionally parts might collide, leading to an invalid state (*i.e.*, an empty set of outputs for certain inputs).

For $j = 1, 2$ let C_j be spatio-temporal components with syntactic interface $(I_j, O_j, D_j, P_j, M_j)$ such that their interfaces are pairwise disjunct, *i.e.*, $X_1 \cap X_2 = \emptyset$ where X is just a substitute for I, O, D, P, M. Their parallel composition denoted by $M_1 \| M_2$ has the syntactic interface

$$\begin{aligned}&(I, O, D, P, M) = \\ &\quad (I_1 \cup I_2, O_1 \cup O_2, D_1 \cup D_2, P_1 \cup P_2, M_1 \cup M_2) \ .\end{aligned}$$

For a specific input $(i, a) \in I \times \mathcal{A}(D)$ and $(o, d, p, m, f) \in (C_1 \| C_2)(i, a)$ the relationship to the inputs and outputs $(o_j, d_j, p_j, m_j, f_j) \in C_j(i_j, a_j)$ is characterized by the following equations.

$$\begin{aligned}\forall j \in \{1, 2\} : \quad & i|_{I_j} {=} i_j, \ o|_{O_j} {=} o_j, \\ & d|_{D_j} {=} d_j, \ p|_{P_j} {=} p_j, \ m|_{M_j} {=} m_j\end{aligned} \tag{1}$$

[2] $\mathbb{B}$ denotes the set of Boolean values.

[3] While there are usually infinitely many valid executions and a single execution may take infinite time, we usually describe these systems by finite notations, such as mathematical functions or modified state machines.

This defines that the input and output, the positions of detectors and parts, and the mover transformations are just evaluated individually by each of the parallel composed components. As shown by the next equation, the forbidden areas are also just combined.

$$\forall t \in \mathbb{N} : \ f.t = f_1.t \sqcup f_2.t \tag{2}$$

Finally the detector activations can be triggered from both the outside of the new component and mutually by the two components, *i.e.*, if a part from one component is detected by (collides with) a detector from the other component.

$$\begin{array}{l} \forall j,k \in \{1,2\}, j \neq k \ \forall \delta \in D_j \ \forall t \in \mathbb{N} : \\ a_j(\delta).t = \left(a(\delta).t \vee \bigvee_{\rho \in P_k} p_k(\rho).t \bowtie d_j(\delta).t \right) \end{array} \tag{3}$$

The important point here is that is the components are strongly causal. This ensures there always exists a unique solution to the equation, as otherwise the component could react to a collision by moving parts such that the collision did not occur in the first place, leading to paradox situations.

In addition to the three equations given before, composition is only valid for an input if the components do not interfere with each other in disallowed ways, *i.e.*, collisions between parts of the two components or penetration of the forbidden volume.

$$\begin{array}{l} \forall t \in \mathbb{N} \ \forall \rho_1 \in P_1, \rho_2 \in P_2 : \\ \quad \neg (p_1(\rho_1).t \bowtie p_2(\rho_2).t \vee \\ \quad\quad f_1.t \bowtie p_2(\rho_2).t \ \vee \ p_1(\rho_1).t \bowtie f_2.t) \end{array} \tag{4}$$

We still allow the forbidden areas to overlap, and parts of a component are allowed to violate its own forbidden area (*e.g.*, $p_1.t \bowtie f_1.t$), as a component is responsible for what may happen in its forbidden area. The contact just states, that no *external* objects may be in this area, as a component may rely on this.

Equipped with these equations we can precisely describe the result of $(C_1 \| C_2)(i,a)$. Denote by R the results according to equations 1 to 3:

$$\begin{array}{l} R := \{(o,d,p,m,f) \in \vec{O} \times \vec{D} \times \vec{P} \times \vec{M} \times V^\infty \ \mid \\ \quad \exists i_1, a_1, o_1, d_1, p_1, m_1, f_1, i_2, a_2, o_2, d_2, p_2, m_2, f_2 : \\ \quad \forall j \in \{1,2\} : \ (o_j, d_j, p_j, m_j, f_j) \in C_j(i_j, a_j) \wedge \\ \quad \text{equations 1 to 3 hold}\} \end{array}$$

If for the construction of any element of R equation 4 was violated, then $(C_1 \| C_2)(i,a) = \emptyset$, otherwise $(C_1 \| C_2)(i,a) = R$. The reason to exclude all outcomes in case of a single possible invalidation is that we do not have control over the composed components, *i.e.*, it is possible to yield an invalid state, so the input may not be used for the composed component.

From the equations it can be seen, that parallel composition is both a commutative and associative operation.

4.4. Constrained activation

It is good engineering practice to explicitly model all effects which contribute to the correct behavior of a system. However, in the parallel composition given previously all possible interactions in terms of detector activations are possible. To allow a more controlled composition, we introduce a constrained activation operation. Let C be a spatio-temporal component with syntactic interface (I, O, D, P, M) and $\rightarrowtail \subseteq P \times D$ a relation describing allowed activations. We intend to only allow a detector δ to be activated by a part ρ, if $\rho \rightarrowtail \delta$.

The *constrained activation of C with respect to $\rightarrowtail$* is denoted by $\text{cact}(C, \rightarrowtail)$ and has the same syntactic interface as C. Its semantics are defined by

$$\text{cact}(C, \rightarrowtail)(i,a) := \begin{cases} \emptyset & \text{if } \exists (o,d,p,m,f) \in C(i,a) \ \exists \rho \in P, \delta \in D \\ & \quad \exists t \in \mathbb{N} : \ p(\rho).t \bowtie d(\delta).t \ \wedge \ \neg(\rho \rightarrowtail \delta) \\ C(i,a) & \text{otherwise} \end{cases}$$

Usually constrained activation will be used in conjunction with parallel composition, so for spatio-temporal components C_j with syntactic interfaces $(I_j, O_j, D_j, P_j, M_j)$ $(j \in \{1,2\})$ and $\rightarrowtail \subseteq (P_1 \cup P_2) \times (D_1 \cup D_2)$ we use the abbreviation $C_1 \|_{\rightarrowtail} C_2$ instead of $\text{cact}(C_1 \| C_2, \rightarrowtail)$.

4.5. Data feedback

With only parallel composition the components could only interact physically (*i.e.*, by collisions between parts and detectors). To support communication using the data input and output ports, we introduce a feedback operation. Let C be a spatio-temporal component with syntactic interface (I, O, D, P, M) and $q : I \rightarrow O$ a *partial* mapping from input to output ports[4] which is *type correct*, *i.e.*, $\forall \iota \in \text{dom}(q) : \ \text{type}_I(\iota) = \text{type}_O(q(\iota))$. Then the *feedback on C according to q* is denoted by $\text{fb}(C, q)$, has syntactic interface (I', O, D, P, M) with $I' := I \setminus \text{dom}(q)$, and is semantically defined by

$$\begin{array}{l} \text{fb}(C,q)(i',a) := \{(o,d,p,m,f) \mid \exists i \in I : \\ \quad i|_{I'} = i' \ \wedge \ (o,d,p,m,f) \in C(i,a) \wedge \\ \quad \forall \iota \in \text{dom}(q) : \ i(\iota) = o(q(\iota))\} \ . \end{array}$$

Due to the strong causality we required for spatio-temporal components, this operation is well defined and the function defined by a component after feedback can be easily constructed by induction on time.

[4] By $\text{dom}(f)$ we denote the domain of a partial function f, *i.e.*, the set of input values for which f is defined.

Feedback is usually used in conjunction with parallel composition to connect multiple components with each other.

4.6. Hiding

Hiding or encapsulation is an engineering practice used to mask irrelevant details, which can be used both to ease understanding of a component and avoid the users of a component to rely on implementation details which might be subject to change. For a spatio-temporal component C with interface (I, O, D, P, M) we can hide outputs or movers by simply erasing them from the interface, thus making them inaccessible. If we want to hide detectors, we have to ensure that when using the component no external objects interfere with those detectors. This can be achieved by extending the forbidden area of the component by the volumes of the hidden sensors. Parts can not be hidden, but their appearance can be made more coarse by treating multiple parts as one larger part. Inputs can not be hidden, because a value has to be assigned to them. However the feedback operation from the previous section removes input ports from the interface, so parallel composition with a component without inputs followed by feedback can be used to hide input ports.

By $H_O \subseteq O$, $H_D \subseteq D$, and $H_M \subseteq M$ we denote the sets of outputs, detectors, and movers we intend to hide. Additionally let $P_1, \ldots, P_n$ be a proper partition of P, *i.e.*, $P_j \neq \emptyset$, $P_j \cap P_k = \emptyset$ for $j \neq k$ and $\bigcup_{j=1}^{n} P_j = P$. Then we denote C after the application of hiding using these sets by $\text{hide}(C, H_O, H_D, \{P_1, \ldots, P_n\}, H_M)$. Its syntactic interface is $(I, O \setminus H_O, D \setminus H_D, \{P_1, \ldots, P_n\}, M \setminus H_M)$ and its semantics are given by

$$
\begin{aligned}
&\text{hide}(C, H_O, H_D, \{P_1, \ldots, P_n\}, H_M)(i, a) = \\
&\quad \{(o|_{O \setminus H_O}, d|_{D \setminus H_D}, p', m|_{M \setminus H_M}, f') \mid \\
&\qquad \exists a'\ \exists (o, d, p, m, f) \in C(i, a')\ \forall t \in Nat : \\
&\qquad f'.t = f.t \sqcup \bigsqcup_{\delta \in H_D} d(\delta).t \wedge \\
&\qquad a'|_{D \setminus H_D} = a\ \wedge\ a'|_{H_D}.t = \text{false} \wedge \\
&\qquad \forall j \in \{1, \ldots, n\} :\ p'(P_j).t = \bigsqcup_{\delta \in P_j} p(\delta).t\} \ .
\end{aligned}
$$

4.7. Positioning

An operation that seems quite obvious when dealing with components in space is a positioning operation, which is used to move a component to its proper place. For this let C be a mechatronic component with syntactic interface (I, O, D, P, M) and $T \in \mathcal{T}^\infty$ a stream of transformations. Then $\text{pos}(C, T)$ denotes C *positioned by* T. The positioned component has the same syntactic interface as C and

$$
\begin{aligned}
&\text{pos}(C, T)(i, a) = \{(o, T(d), T(p), m', T(f)) \mid \\
&\quad \exists (o, d, p, m, f) \in M(i, c)\ \forall t \in \mathbb{N} :\ m'.t = T.t \circ m.t\}
\end{aligned}
$$

where T is interpreted point-wise on V^∞ and $\vec{D}$ resp. $\vec{P}$.

4.8. Connecting movers

Finally we want to express that a component is connected to another one (like for example a robotic gripper is connected to a robotic arm) via some mover and thus has to follow every movement.

For $j = 1, 2$ let C_j be spatio-temporal components with syntactic interface $(I_j, O_j, D_j, P_j, M_j)$ and $\mu \in M_1$ a mover. Then the connection of M_2 to M_1 via the mover μ (written as $M_1 \overset{\mu}{\circ\!\!-} M_2$) is defined as

$$
M_1 \overset{\mu}{\circ\!\!-} M_2 := M_1 \parallel \text{pos}(M_2, \pi_\mu(M_1)) \ ,
$$

where π_μ is the projection to the corresponding transformation of μ.

As a connection by a mover results in a single component, no cyclic connections can occur. Thus the connections by movers form a forest in the components, which corresponds to serial kinematics.

4.9. Implications

The model presented so far has some interesting properties, which are summarized in this section. The first one is, that all operations either only affect the spatial parts (parallel composition, mover connection) or the communication ports (feedback). Thus the interplay between logic data and spatial effects (motion and collisions) has to happen in the atomic components. An example for this is given in the next section, where a sensor is modeled as a component which converts detector activations to data sent on the output port.

The other property is that contrary to other formalisms the partiality of the specification (the semantic interface function) is not used to express under-specification, but rather to express invalid input. Different from pure software systems such invalid input could lead to output that is not only undefined but might describe an invalid spatial configuration that might be impossible or hazardous for the real system. While for components it can be part of their contract to not work for every possible input, the entire system usually is expected to have a defined (valid) behavior for any input, as the environment is out of control of the system and thus any input has to be dealt with in a non-destructive way. This totalization is usually performed by reducing the possible inputs, which can be achieved by hiding detectors or filtering inputs using an additional component.

5. Example

The presented formalism is not intended to be used to actually model systems, but rather as a means for reasoning about them. The modeling should be performed using some technique which is based on the semantics presented here.

Nonetheless, we give an exemplary description of a system in this formalism to show how the presented concepts are applied. For all examples we assume the three-dimensional Euclidean space given as an example in Sec. 3.

5.1. Industrial robot

Our first example is a simplified industrial robot, which is shown in Fig. 1. It consists of three parts: a base which can be rotated, the arm, which can be extended, and the gripper.

We model the base by a component C_{base} with syntactic interface $(\{i_{\text{base}}\}, \{o_{\text{base}}\}, \emptyset, \{p_{\text{base}}\}, \{m_{\text{base}}\})$. Via the input port, commands for rotation of the base are sent, while on the output port it reports its current orientation in degree, thus $\text{car}(\text{type}(i_{\text{base}})) = \{\text{left}, \text{right}, \text{halt}\}$ and $\text{car}(\text{type}(o_{\text{base}})) = \mathbb{Z}$. The base behaves deterministically and is defined by $C_{\text{base}}(i) = \{(o, p, m, f)\}$, where o, p, m, f are constrained by ($t \in \mathbb{N}$)

$$o.0 = 0 \ ,$$

$$o.(t{+}1) = \begin{cases} \max\{(o.t) - 1, -90\} & \text{if } i.t = \text{left} \\ \min\{(o.t) + 1, 90\} & \text{if } i.t = \text{right} \\ o.t & \text{otherwise} \end{cases} ,$$

$p = p_0^\infty$, where p_0 is the volume corresponding to the base,

$$m.t = \text{“rotation by } o.t \text{ degree”},$$

$$f = 0_V^\infty \ .$$

So the base may rotate between -90 and 90 degree, but its shape is always the same.

The arm is described by the component C_{arm} with syntactic interface $(\{i_{\text{arm}}\}, \{o_{\text{arm}}\}, \emptyset, \{p_{\text{arm}}\}, \{m_{\text{arm}}\})$. Via the input port, commands for extending the arm are sent, while on the output port it reports its current length in centimeters, thus $\text{car}(\text{type}(i_{\text{arm}})) = \{\text{extend}, \text{retract}, \text{halt}\}$ and $\text{car}(\text{type}(o_{\text{arm}})) = \mathbb{N}$. The arm behaves deterministically and is defined by $C_{\text{arm}}(i) = \{(o, p, m, f)\}$, where o, p, m, f are constrained by ($t \in \mathbb{N}$)

$$o.0 = 50 \ ,$$

$$o.(t{+}1) = \begin{cases} \min\{(o.t) + 1, 100\} & \text{if } i.t = \text{extends} \\ \max\{(o.t) - 1, 50\} & \text{if } i.t = \text{retract} \\ o.t & \text{otherwise} \end{cases} ,$$

$p.t$ is the corresponding volume with the arm extended to a length of $o.t$ centimeters,

$$m.t = \text{“translation by } o.t - 50 \text{ centimeters”},$$

$$f = 0_V^\infty \ .$$

The arm may be extended from 50 to 100 centimeters. Note that we do not model how this is achieved technically, but only how this affects the shape of the arm and the translation it applies to connected components.

The component C_{gripper} for the gripper is not given here, as it is similar to those given before. The entire robot can be expressed as

$$C_{\text{robot}} = (C_{\text{base}} \overset{m_{\text{base}}}{\circ\!\!-} C_{\text{arm}}) \overset{m_{\text{arm}}}{\circ\!\!-} C_{\text{gripper}} \ .$$

5.2. Interacting robots

As a second example we model two robots which interact in unloading a container which can be placed between them. A sketch is shown in Fig. 2 which also depicts the sensor which is needed to detect the presence of a container. For this we use two components C_{robot1} and C_{robot2}, which are the same as the robot given in the previous example, but with the numbers 1 and 2 attached to all associated names[5]. Additionally we need a component C_{sensor} with syntactic interface $(\emptyset, \{o_{\text{sensor}}\}, \{d_{\text{sensor}}\}, \emptyset, \emptyset)$ characterized by $C_{\text{sensor}}(a) = \{(o, d\}$,where $\text{car}(\text{type}(o_{\text{sensor}})) = \mathbb{B}$, $o.0 = \text{false}$, $o.(t{+}1) = a.t$, and d is the stream giving the position of the sensor as shown in the figure. So the sensor is just used for converting spatial collision activations to logical data signals. The electro-mechanical part of the system is thus captured by

$$C_{\text{unload}} = C_{\text{sensor}} \parallel_{\rightarrowtail} (C_{\text{robot1}} \parallel \text{pos}(C_{\text{robot2}}, T_{\text{robot2}}^\infty)) \ ,$$

where T_{robot2} describes the transformation used to place the second robot to its intended position and $\rightarrowtail$ is the empty relation as we do not want the robots to activate the sensor.

Currently all inputs and outputs are externally accessible, allowing the robots to be moved around. In this setup however, there obviously is the possibility of the robots' arms to collide causing damage. For example for any input with $i_{\text{arm1}} = i_{\text{arm2}} = \text{extend}^\infty$ the result would be the empty set indicating an invalid state. One possible solution is the inclusion of a controller component C_{ctl}, which takes high-level commands and applies them to the robots, but at the

[5] Formally this could be captured by a renaming operation, however this kind of reuse should be handled by a modeling environment and thus we do not deal with it here.

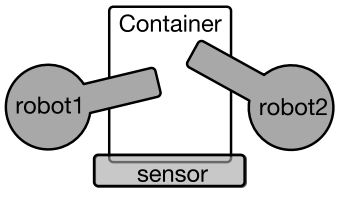

Figure 2. Sketch of the setup with two robots as seen from the top.

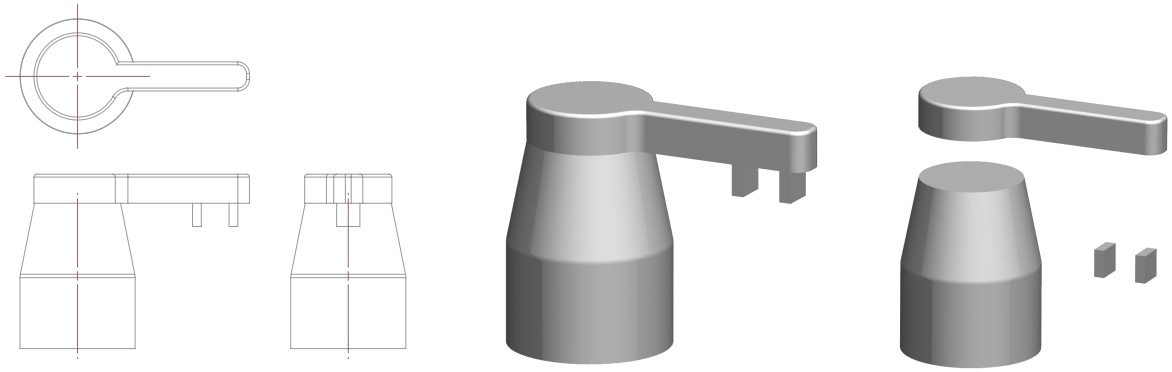

Figure 1. Several views of the robot: draft, 3d view, exploded view.

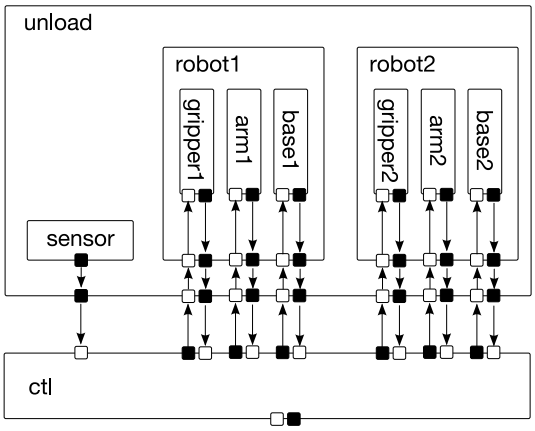

Figure 3. Component diagram for the two robots.

same time tries to avoid collisions between them. It has the syntactic interface

$$(\{i_{\text{cmd}}, i_{\text{sensor}}, i_{\text{r1b}}, i_{\text{r2b}}, i_{\text{r1a}}, i_{\text{r2a}}\}, \{o_{\text{status}}, o_{\text{r1b}}, o_{\text{r2b}}, o_{\text{r1a}}, o_{\text{r2a}}\}, \emptyset, \emptyset, \emptyset) \ .$$

We define the partial connection relation q by the pairs

$$\{(i_{\text{r1b}}, o_{\text{base1}}), (i_{\text{r1a}}, o_{\text{arm1}}), \ldots, (i_{\text{base2}}, o_{\text{r2b}}), (i_{\text{arm2}}, o_{\text{r2a}}), (i_{\text{sensor}}, o_{\text{sensor}})\} \ .$$

A semantic interface for C_{ctl} is not provided here, but obviously the goal should be to define it in a way that the entire system $C_{\text{system}} = \text{fb}(C_{\text{ctl}} \| C_{\text{unload}}, q)$ specifies a total function, *i.e.*, there is no input leading to an invalid state (indicated by missing output).

As a summary the component compositions are graphically depicted in Fig. 3, which nicely shows the hierarchical decomposition of the complex system to manageable components. The input and output ports of components are drawn as white and black little squares and the arrows indicate the data flow through the system. For the diagram we also added ports/channels for the grippers, which were not discussed before.

6. Properties of spatio-temporal systems

In this section we discuss properties for instances of our spatio-temporal model. We only give a logic characterization of the properties here. Checking for these properties is beyond the scope of this paper and can probably only be solved for a reduced set of components and systems, as most of the required checks are undecidable in general.

6.1. Spatial compatibility

Two spatio-temporal components C_1, C_2 with syntactic interface $(I_j, O_j, D_j, P_j, M_j)\ (j = 1, 2)$ are called *spatially compatible*, if for all $i \in \overrightarrow{I_1 \cup I_2}$ and $a \in \mathcal{A}(D_1 \cup D_2)$:

$$C_1(i|_{I_1}, a|_{D_1}) \neq \emptyset \ \wedge \ C_2(i|_{I_2}, a|_{D_2}) \neq \emptyset \ \Rightarrow \\ (C_1 \| C_2)(i, a) \neq \emptyset \ .$$

This encodes that the combined components may not run into an invalid state for inputs which would not cause problems with both components in isolation.

A sufficient condition for spatial compatibility is *spatial separation*, which intuitively captures that two components are far enough from each other to not affect each other. For a component C we call $B \in V$ a (not necessarily unique) *bounding volume*, if

$$\forall i \in \vec{I}, a \in \mathcal{A}(D)\ \forall (o, d, p, m, f) \in C(i, a)\ \forall t \in \mathbb{N} : \\ \forall \delta \in D :\ d(\delta).t \sqsubseteq B \ \wedge \\ \forall \rho \in P :\ p(\rho).t \sqsubseteq B \ \wedge \\ f.t \sqsubseteq B \ .$$

We call C_1 and C_2 spatially separated, if there are bounding volumes B_1 for C_1 and B_2 for C_2 such that $\neg(B_1 \bowtie B_2)$.

6.2. Refinement

The idea of refinement is to gradually enrich a model with more details, thereby ensuring (proving) that each refined model does not violate the guarantees provided by the coarser model. The goal usually is to perform refinement far enough to result in an implementation of the system, which is then *correct by construction*.

Depending on the kind of details added, different notions of refinement can be derived. We will concentrate on behavioral refinement here, which basically reduces nondeterminism. Let C_1 and C_2 be two spatio-temporal components with the same syntactic interface (I, O, D, P, M). We call C_2 a refinement of C_1 (denoted by $C_1 \rightsquigarrow C_2$), if for all $(i, a) \in \vec{I} \times \mathcal{A}(D)$

$$C_1(i, a) \neq \emptyset \;\Rightarrow\; C_2(i, a) \subseteq C_1(i, a) \;\; .$$

So whenever an input is invalid for C_1, we do not care about the results of C_2; otherwise C_2 may not behave in a way that was not already possible for C_1.

Given the definitions in Sec. 4, behavioral refinement has some nice properties, namely that for all components C_1, C_2, C_3 and "matching" q and T the following statements hold:

$$\begin{array}{lcl} C_1 \rightsquigarrow C_2 & \Rightarrow & C_1 \| C_3 \rightsquigarrow C_2 \| C_3 \\ C_1 \rightsquigarrow C_2 & \Rightarrow & \mathrm{fb}(C_1, q) \rightsquigarrow \mathrm{fb}(C_2, q) \\ C_1 \rightsquigarrow C_2 & \Rightarrow & \mathrm{pos}(C_1, T) \rightsquigarrow \mathrm{pos}(C_2, T) \end{array}$$

Similar rules are also valid for hiding, constrained activation, and mover connection. This allows us to perform refinement on the individual atomic components and as a result get a refinement of the overall system.

7. Future work

Future work can be roughly divided into three areas. The first one is the extension of the semantic model to also capture the generation and transformation of passive objects. Those passive objects do not themselves posses a behavior, but rather are affected by the individual components of a system, which may pass these objects to each other and change their location in space. Such an extension could be used for modeling material transportation and manipulation, which is one of the core tasks of systems in the industrial automation domain.

The second area of further research is the description of practically usable modeling techniques based on this semantic model. This includes suitable techniques for representing atomic components (like some variant of automata) as well as an intuitive (possibly graphical) syntax for the composition of components. A first step in this direction has been made in [14], but the link to the model presented here still has to be made. Along this line we also look into building suitable tool support for the creation of those models, as modeling systems of realistic size is not feasible without them, and automatic analysis techniques can hardly be applied to models written on paper. First results are available as part of AF/STEM[6] (Fig. 4), which is a branch of AutoFOCUS [2, 18], a prototypical tool adaption of the FOCUS theory.

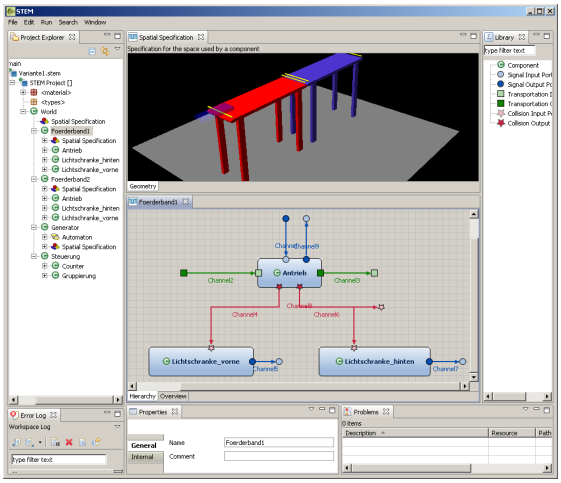

Figure 4. Screen shot of the prototypical modeling tool AF/STEM.

Finally, the various uses of such a spatio-temporal system model should be subject to further research. This includes advanced analysis and verification techniques to prove these systems correct with respect to some specification, which in turn raises the question of suitable formalisms for the specification of such systems. Another application currently investigated in the context of the AutoVIBN project is the generation of (partial) development artifacts, such as coarse CAD models, source code, or simulation models used for testing. This assumes that the creation of a suitable system model happens early in the development process.

8. Conclusion

In this paper we presented a semantically founded model for the description of computer-based spatio-temporal systems. The model is based on a decomposition into hierarchical components, and the description of component behavior by a relation on streams which represent the data inputs and outputs, activations, shapes, and positions of the component over time. The expressiveness of the model was demonstrated by two examples and we showed how several

[6] Details at `http://af3.in.tum.de/index.php/STEM`.

properties of systems can be formalized in this framework. We perceive this as a solid foundation for further research in the area of modeling and analyzing spatio-temporal and mechatronic systems.

References

[1] M. Broy. Refinement of time. *Theoretical Computer Science*, 253(1):3–26, 2001.

[2] M. Broy, F. Huber, and B. Schätz. AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. *Informatik Forschung und Entwicklung*, 13(13):121–134, 1999.

[3] M. Broy, I. Krüger, and M. Meisinger. A formal model of services. *ACM Transactions on Software Engineering Methodology*, 16(1), 2007.

[4] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[5] P. J. L. Cuijpers and M. A. Reniers. Hybrid process algebra. *Journal of Logic and Algebraic Programming*, 62(2):191–245, 2005.

[6] B. A. El-Geresy. The space algebra: Spatial reasoning without composition tables. In *ICTAI'97: Proceedings of the 9th International Conference on Tools with Artificial Intelligence*, pages 67–74, 1997.

[7] V. Engelson. *Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing*. PhD thesis, Linköpings Universitet, 2000.

[8] D. Gabelaia, R. Kontchakov, Á. Kurucz, F. Wolter, and M. Zakharyaschev. Combining spatial and temporal logics: Expressiveness vs. complexity. *Journal of Artificial Intelligence Research*, 23:167–243, 2005.

[9] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[10] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[11] T. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, NATO ASI Series F: Computer and Systems Sciences 170, pages 265–292. Springer, 2000.

[12] C. M. Hoffmann. *Geometric and solid modeling*. Morgan Kaufmann, 1993.

[13] J. Huang, J. Voeten, M. A. Groothuis, J. F. Broenink, and H. Corporaal. A model-driven design approach for mechatronic systems. In *ACSD'07: Proceedings of the 7th International Conference on Application of Concurrency to System Design*, pages 127–136. IEEE, 2007.

[14] B. Hummel and P. Braun. Towards an integrated system model for testing and verification of automation machines. In *MiSE '08: Proceedings of the 2008 international workshop on Models in Software Engineering*, pages 51–56. ACM, 2008.

[15] O. Kutz, F. Wolter, H. Sturm, N.-Y. Suzuki, and M. Zakharyaschev. Logics of metric spaces. *ACM Transactions on Computational Logic*, 4(2):260–294, 2003.

[16] N. Lynch, R. Segala, and F. Vaandraager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

[17] Object Management Group. OMG SysML specification v. 1.0, May 2006.

[18] B. Schätz, A. Pretschner, F. Huber, and J. Philipps. Model-based development of embedded systems. In *OOIS Workshops*, pages 298–312, 2002.

[19] P. Struss, A. Kather, D. Schneider, and T. Voigt. A compositional mathematical model of machines transporting rigid objects. In *ECAI'08: Proceedings of the 18th European Conference on Artificial Intelligence*, 2008.

[20] M. Tiller. *Introduction to Physical Modeling with Modelica*. Springer, 2001.