University of Pennsylvania
ScholarlyCommons

Departmental Papers (CIS)   Department of Computer & Information Science

March 2006

R-Charon, a Modeling Language for Reconfigurable Hybrid Systems

Fabian Kratz
Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

George Pappas
University of Pennsylvania, pappasg@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_papers

R-Charon, a Modeling Language for Reconfigurable Hybrid Systems

Abstract

This paper describes the modeling language as an extension for architectural reconfiguration to the existing distributed hybrid system modeling language Charon. The target application domain of R-Charon includes but is not limited to modular reconfigurable robots and large-scale transportation systems. While largely leaving the Charon syntax and semantics intact, R-Charon allows dynamic creation and destruction of components (agents) as well as of links (references) between the agents. As such, R-Charon is the first formal, hybrid automata based modeling language which also addresses dynamic reconfiguration. We develop and present the syntax and operational semantics for R-Charon on three levels: behavior (modes), structure (agents) and configuration (system).

R-Charon, a Modeling Language for Reconfigurable Hybrid Systems⋆

Fabian Kratz[1], Oleg Sokolsky[2], George J. Pappas[2], and Insup Lee[2]

[1] Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands
[2] University of Pennsylvania, Philadelphia, USA

Abstract. This paper describes the modeling language R-Charon as an extension for architectural reconfiguration to the existing distributed hybrid system modeling language Charon. The target application domain of R-Charon includes but is not limited to modular reconfigurable robots and large-scale transportation systems. While largely leaving the Charon syntax and semantics intact, R-Charon allows dynamic creation and destruction of components (agents) as well as of links (references) between the agents. As such, R-Charon is the first formal, hybrid automata based modeling language which also addresses dynamic reconfiguration. We develop and present the syntax and operational semantics for R-Charon on three levels: behavior (modes), structure (agents) and configuration (system).

1 Introduction

A hybrid system typically consists of a collection of components interacting with each other and with an analog environment. In many real world systems, the collection of components as well as the components they interact with may change dynamically, i.e., reconfigure [1].

In the world of software design the concept of reconfiguration is well recognized. Object orientation is (becoming) the main design and implementation paradigm. Creation and destruction of objects as well as changing the communication structure of the objects are at the core of the object oriented design paradigm.

Traditional object oriented design methodologies and languages, however, only support the modeling of discrete systems. Despite the growth of hybrid modeling languages [2–6], most hybrid modeling languages do not support reconfiguration. To properly describe and analyze reconfigurable hybrid systems, a formal approach is necessary which integrates reconfigurable discrete behaviors with continuous behaviors. In this paper we present a reconfiguration extension for the hierarchical hybrid modeling language Charon [2, 3].

Charon is a hybrid modeling language with support for architectural as well as behavioral hierarchy. The building block for describing a system architecture

---

is an *agent*, which can communicate with other agents. Concurrency of agents and hiding of information is provided by a composition and a hiding operator. The building block for describing behavior in an agent is a *mode*. A mode is a hierarchical hybrid state machine, i.e., it can have submodes and transitions connecting them. An agent alternates between taking a discrete and a continuous step. A discrete step consists of a series of transitions leading from the currently active atomic mode, to another atomic mode. This flow of control is determined by mode invariants, transition guards, and transition actions possibly changing mode variables. A continuous step amounts to passage of time, during which the continuous variables evolve according to the algebraic and differential constraints of the active modes.

There could be many notions of (re)configuration for hybrid systems. We focus on reconfiguration in two, in a sense similar, application domains: large-scale transportation systems and modular reconfigurable robots. A transportation system typically consists of a large number of possibly mobile entities, competing for bounded resources. These entities can enter and leave an environment dynamically. Furthermore, (groups of) entities nearing each other may dynamically set up a communication connection to prevent a collision or to continue as a group to allow for a more efficient use of the resources. Examples of large-scale transportation systems include highway control systems [7], unmanned aerial vehicles [8], and air traffic control systems [9]. A modular robot is built up from homogeneous modules which can be connected to each other [10, 11]. Typically there are only a few different types of modules, where each type is designed to be very orthogonal with respect to the connection to other modules. In this way a number of many relative simple modules can be connected to form a sophisticated robot. A comprehensive overview of the different existing modular robot systems can be found in Chapter 4 of [12].

The main contribution of this paper is the formal definition of R-Charon and its features. In addition to the Charon features, our extension supports agent creation and destruction as well as dynamic communication connections between the agents. This makes R-Charon the first formal, hybrid automata based modeling language with explicit support for reconfiguration. We used two guidelines in the design of R-Charon: minimize the amount of changes to Charon and minimize the number of restrictions on the use of the syntax. Note that the latter comes at the expense of more sophisticated semantics.

Related work. An early approach to hierarchical hybrid modeling with support for reconfiguration is SHIFT [13]. R-Charon is inspired by its features while enjoying the formal Charon semantics. The Φ-calculus [6] is a process algebraic based hybrid reconfigurable modeling language. As an extension of Milner's π-calculus [14] it inherits the powerful reconfiguration primitives on process algebra terms. However, as pointed out in [4] the Φ-calculus considers continuous behavior to be a property of an explicit environment instead of being part of an agent as we do. Furthermore a process algebraic approach has the disadvantage that it is difficult to learn and use due to some of its technicalities [15]. Besides a hybrid extension to I/O Automata [5], also a reconfiguration extension [16]

exists, though not both are combined into a single framework. Some work has been done on reconfiguration in discrete state machines [17] for programmable hardware. The state machines are reconfigured by adding and removing states and transitions, i.e., take place at the behaviorial level in contrast to the architectural level we aim at.

2   Reconfiguration

Before we present R-Charon, we first formalize the notion of reconfiguration we use. The definition is based on the reconfiguration possibilities of modular robots and large-scale transportation systems, and is inspired by SHIFT [13].

A model of a system consists of a set of components C. Each component $c \in C$ consists of a single set of links L, containing links to other components to which the component is either logically or physically connected. The set of links L of a specific component c is denoted by $c.L$. A component c having at least one link to a component d, means that c can communicate with d, where we consider linking not to be reciprocal.

Given a system with a set of components C, the reconfiguration primitives given below can take place. More complex operations can be performed by a series of primitives. Since a reconfiguration-only view is presented, the time instant or the event at which the reconfiguration happens is not relevant.

1. **Adding a component**: A component $c \in C$ can create a new component c_{new} and add it to the set C, i.e., $C := C \cup \{c_{new}\}$. As a consequence components can now link to c_{new}. We assume that c_{new} is of a certain type, which is known beforehand and defines the structure of the new component.
2. **Removing a component**: A component $c \in C$ can remove an arbitrary $c_r \in C$ from the set C, i.e., $C := C - \{c_r\}$.
3. **Adding a link**: A component $c \in C$ can add an arbitrary $c_a \in C$ to its set $c.L$, i.e., $c.L := c.L \cup \{c_a\}$. As a consequence component c can now communicate with c_a.
4. **Removing a link**: A component $c \in C$ can remove an arbitrary $c_r \in c.L$ from $c.L$, i.e., $c.L := c.L - \{c_r\}$.

The configuration of the world is determined by the set of components C and the specific values of the sets of links L of all components. To keep a system consistent after removal of a component, all links to the removed component are removed as well.

3   Application Example

In this section we present an application example that exhibits the new features of R-Charon. In the course of the example, we introduce some graphical R-Charon syntax, and point out the difficulties in defining the semantics for R-Charon. Our example is inspired by next generation air-traffic control supporting

4

free flight for commercial airplanes [9], which allows airplanes to navigate themselves to their target with minimal air-traffic control interaction. We focus on a section of airspace (*center*) in which airplanes enter and leave, see Figure 1. The center has a designated no-fly zone, e.g. a military training operation area off limit to commercial airplanes. In case an airplane approaches the designated no-fly zone, ground control takes over the navigation of the airplane by giving way-points, directing the airplane around the no-fly zone. Collision avoidance is not considered in this example.
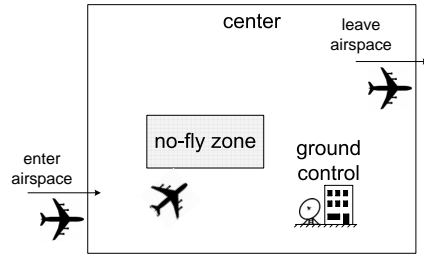

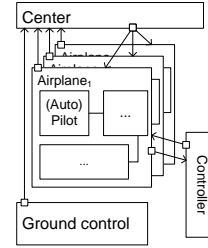**Fig. 1.** Air-traffic control example

Fig. 2. Configuration snapshot

Figure 2 presents a snapshot of the hierarchical agents and the configuration of the system. The arrows originating from a white box depict the links from one agent to another. The *Center* agent represents the section of the airspace and stores links to all airplanes in the airspace. Airplanes entering and leaving the center are modeled by creation and destruction of *Airplane* agents. The *Ground control* agent monitors the airspace in the center. In case an *Airplane* agent approaches the no-fly zone, the *Ground control* agents creates a new controller agent. The *Controller* agent contacts the corresponding airplane and guides it around the no-fly zone. As in Charon, each agent consists of one or more
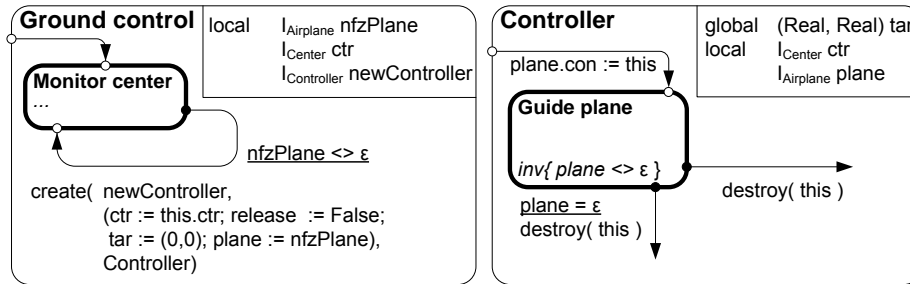

Fig. 3. Top-level modes of a *Ground control* and *Controller* agent

top-level modes, which can contain submodes. Figures 3 and 4 depict simplified

views of the top-level modes of a *Ground control* agent, a *Controller* agent, and a number of *Airplane* agents, respectively. Modes not specified in detail are marked with a fat line. Assume that in the mode *Monitor center*, an airplane approaching the no-fly zone will be assigned to the *nfzPlane* reference variable. This triggers the creation of a new controller referred to by the *newController* variable. In the discrete initialization step of a *Controller* agent, the no-fly zone violating airplane is notified by setting a reference of the *Airplane* agent to itself. Note that this reference represents a link from *Airplane* to *Controller*. The airplane switches to *Ground control guided* mode and follows a target coordinate given by the controller, which is computed in the *Guide plane* mode. As soon as either the airplane leaves the center or the controller decides that the airplane has maneuvered successfully around the no-fly zone, the controller is destroyed. In the latter case, the airplane switches back to the *Free flight* mode.
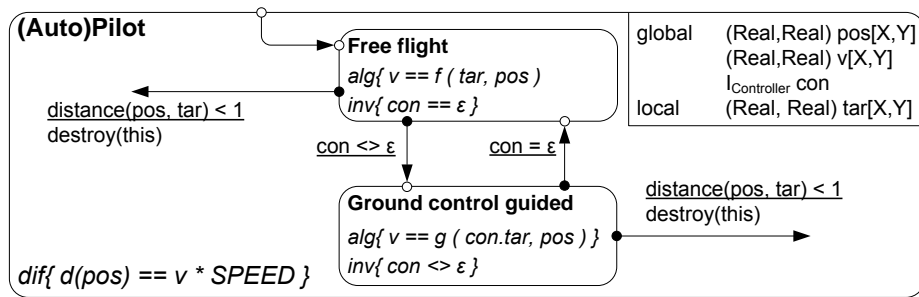

Fig. 4. Top-level mode of an airplane agent

The most prominent semantic difficulties are related to agent creation and destruction. With agent creation the question arises when new agents take their discrete initialization step. Moreover, agent creation during initialization of a created agent leads to questions about creation and initialization order. With agent deletion the question arises how and when to update affected reference variables to reflect the deletion. The common difficulty lies in the compositional and hierarchial structure of Charon and the fact that creation or destruction is an action of a mode that has an effect on the much higher system level.

4 R-Charon

4.1 Notation

Let T be a tuple $(t_1, t_2, \ldots, t_n)$. The ith element of T is identified by $T.t_i$. In other words, the tuple element-names are used as record names and the period is used as a selector operator. This notation is extended to sets of tuples as follows. Let ST be a set of tuples with the same structure. The shorthand notation $ST.t_i$ with t_i a set will be used for $\bigcup_{T \in ST} T.t_i$.

Let V be a set of typed variables. A *valuation* for V is a function mapping from V to values, where the mapping is assumed to be type correct. The set of valuations over V is denoted $\mathcal{Q}_V$. Restriction of a valuation $q \in \mathcal{Q}_V$ to a set of variables $W \subseteq V$ is denoted as $q[W]$. Function application of a valuation $q \in \mathcal{Q}_V$ to a variable $v \in V$ is written as $q(v)$ and returns the value of the variable v.

A *flow* for a set V of variables is a differentiable function f from a closed interval of non-negative reals to the set of valuations: $f : [0, \delta] \to \mathcal{Q}_V$, with $\delta \geq 0$ the *duration* of the flow. The set of all flows for V is denoted as $\mathcal{F}_V$. Restriction of a flow $f \in \mathcal{F}_V$ to a set of variables $W \subseteq V$ is denoted as $f[W]$.

A list l with elements $a_1, \ldots, a_n$ is written as $\langle a_1, \ldots, a_n \rangle$. A list with zero elements is written as $\langle \rangle$. Concatenation of two lists $l_a = \langle a_1, \ldots, a_n \rangle, l_b = \langle b_1, \ldots, b_m \rangle$ is denoted by $l_a \frown l_b$ and results in the list $\langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle$.

4.2 Syntax

The syntax of Charon is extended to accommodate the proposed (re)configuration concept. The syntax is presented in a top-down fashion. Although counterintuitive for a modular modeling language, this approach is more suitable since the reconfiguration infrastructure is defined on the higher system and agent levels and used in the lower mode level.

System. The components of Section 2 are mapped onto Charon agents. Consequently, agents can be created and destroyed. Moreover, when creating new agents dynamically, the structure of the agent has to be known beforehand. To capture the dynamic set of agents and the possible structures of new agents, we define an R-Charon system as:

Definition 1 (System). *An R-Charon system is a tuple $(\mathcal{S}, \mathcal{A})$, where $\mathcal{S}$ is a set of structures and $\mathcal{A}$ is a set of parallelly composed agents. Each agent is an instantiation of a structure from the set $\mathcal{S}$.*

Structures and Agents. Assume that a system $Sys = (\mathcal{S}, \mathcal{A})$ is given. A structure is then defined as:

Definition 2 (Structure). *A structure $S \in \mathcal{S}$ is a tuple (TM, V), where TM is a set of top-level modes and V is a set of typed variables.*

A structure is a blueprint for agents. The set of top-level modes consists of R-Charon modes, which are Charon modes extended as described in the mode syntax section further below. The top-level modes of the structure collectively define the behavior of each R-Charon agent which is an instance of the structure. The set of variables V is partitioned into two sets: a set of local variables V_l and a set of global variables V_g. All global variables of the structure have to originate in some top-level mode, i.e., $V_g \subseteq TM.V_g$. As in Charon, a variable can be of any type, as long as it has a type correct valuation.

To facilitate the concept of adding and removing links between agents, we introduce reference variables. Each instantiated reference variable represents a

link to an agent. This introduces another partitioning of the set of variables V of a structure into two sets: the set of reference variables variables V_r and the set of non-reference variables V_{nr}. A reference variable which is instantiated to point to an agent A can be used to access the global variables of A. Note that a reference variable can be a global or a local variable.

A straightforward choice for the type of reference variables would be any $S \in \mathcal{S}$. However, to allow for a greater flexibility we introduce and use the notion of *interface*. The interface of an Agent A of structure S is defined to be the set of global variables of S, $S.V_g$. The set of interfaces $\mathcal{I}$ is then $\mathcal{I} = \{S.V_g | S \in \mathcal{S}\}$. The type of a reference variable can be any $I \in \mathcal{I}$, possibly representing a link to an agent with a *compatible* interface J. An agent with interface I_1 is compatible to a reference variable of type interface I_2, if all variables available in I_2 are also present in I_1, i.e., $I_2 \subseteq I_1$. These definitions enable interface specialization, allowing a single reference variable to link to agents with compatible interfaces but different behaviors.

Each agent $A \in \mathcal{A}$ is an instantiation of a certain structure $S \in \mathcal{S}$, i.e., there exists a function $s : \mathcal{A} \rightarrow \mathcal{S}$ mapping each agent to a structure. The structure of each agent remains fixed throughout its entire lifespan. An agent is defined as :

Definition 3 (Agent). *An agent $A \in \mathcal{A}$ of structure $s(A)$ is a tuple (TM, V, I), where TM is a set of top-level modes, V a set of variables, and I is a set of possible initialization assignments to the variables of A.*

The set of top-level modes is a copy of the set of top-level modes of the structure of the agent. The set of variables is a copy of the set of variables of the structure of the agent. The set V is extended with a special fixed variable *this*, always referring to the agent itself. The use of this variable will become apparent in the mode syntax presented below.

Modes. The high-level definition of the mode is identical to a Charon mode:

Definition 4 (Mode). *A mode M is a tuple $(E, X, V, SM, Cons, T)$, where E is a set of entry control points, X is a set of exit control points, V is a set of variables, SM is a set of submodes, $Cons$ is a set of constraints, and T is a set of transitions.*

The entry and exit control points and the submodes are the same as in Charon modes. A mode M is called *atomic* if $M.SM = \emptyset$ and *composite* otherwise. The syntax of the sets of variables, constraints, and transitions is extended to facilitate reconfiguration.

In R-Charon the configuration of the agents and the system can change as the system evolves. Hence, the Charon concept of using fixed agent input and output variables in the description of a mode is changed. Now modes can use the global variables of all agents to which they have a reference. To prevent undesirable behavior, all global variables of agents except for the (discrete) global reference variables are defined to be single writer variables, i.e., read-only for the modes in other agents. Allowing to write to global reference variables provides more flexibility in modeling reconfiguration.

To use the global variables of referenced agents in a mode, some additional syntax is introduced. Consider an agent A with the global variables $A.V_g$. Any mode in an agent with a reference variable v_r pointing to A, can use any global variable $v \in A.V_g$ of A by putting $v_r.v$ in its constraint or transition definitions.

The set of variables V of a mode is partitioned into subsets V_a and V_d, the sets of analog and discrete variables, respectively. In addition, V is partitioned into subsets V_g and V_l of global and local variables, respectively. Let $V_{ref} \subseteq V_d$ be the subset of V containing all reference variables of the mode. Define V_+ to be the set of the readable variables, i.e., the union of V and the sets of global variables V_g of the interfaces of the reference variables in V_{ref}. Moreover define the set V_{act} to be the set of writable variables, i.e., the union of V and the subsets of global reference variables of the interfaces of the reference variables in V_{ref}. The set of derivatives of the variables in V is denoted as $d(V)$.

To enable creation and destruction of agents, the syntax of the action part of the transitions is extended with two special operations: *create* and *destroy*. The create operation has a three tuple argument of the form $(v_r, Init, S)$. With v_r a reference variable afterwards pointing to the new agent, $Init \in \mathcal{Q}_{S.V}$ an initialization of the variables of the agent, and S the structure of the agent to be created. Similar to assignments to reference variables, the interface of S is required to be compatible to the interface type of v_r. The destroy action has one argument, a reference variable pointing to the agent that has to be destructed. Note that self-destruction of an agent is possible using the *this*-variable.

As already formalized in the Charon semantics, T is a set of transitions of the form (e, α, x), where $e \in E \cup SM.X$ and $x \in X \cup SM.E$. The action α has a guard γ attached to it, which in turn is a predicate over the set of valuations of V_+, $\mathcal{Q}_{V_+}$. The action α is a sequence of assignments to the variables V_{act} or create or destroy statements.

Each assignment is of the form $x = g(x_1, \ldots, x_n)$ with $x \in V_{act}$ and x_1, $\ldots$, $x_n \in V_+$. The function g might be any function on the given arguments, which returns a value with the same type as x. Note that by the definition of V_{act} new links to agents can be created either through assignments to reference variables of the mode or through assignments to global reference variables of referenced agents. A reference to an agent can be removed by assigning a special value ϵ to the reference variable representing the link. Adding as well as removing a link are demonstrated in the *Controller* mode of Figure 3.

As in Charon, the set of constraints consists of a set of invariants, a set of algebraic constraints, and a set of differential constraints, which together define the flow permitted in the mode. Similar to transition actions, the right-hand side of constraints can include variables from the set of global variables of referenced agents, V_+.

4.3 Informal Semantics

The extensions to Charon consist of two parts. The first part consists of the reference variables and the use of global variables of referenced agents in the mode constraints and transitions. This combination enables creation and destruction

of links. The second part consists of the creation and destruction of agents. The semantics are defined on three levels: mode, agent, and system. An extensive discussion on the informal semantics and the motivated choices made, can be found in Section 4.4 of [18].

Upon agent creation, a new agent of the specified structure will be created and initialized according to the given initialization assignment on the mode level. The reference variable passed along as a parameter of the create command will point to the newly created agent. On the system level, the newly created agent will be added to the set of agents of the system.

Upon agent destruction, the agent referred to by the argument of the destroy operation, is removed from the set of agents of the system. The passed reference variable is set to ϵ. Upon deleting an agent, all reference variables in the system referring to the deleted agent are set to ϵ.

4.4 Semantics

Mode Operational Semantics. The set of all variables of a mode M as well as all variables of the submodes of M is defined recursively as $M.V_* = M.V \cup M.SM.V_*$. A subset of $M.V_*$ containing all reference variables of a mode M and its submodes, is defined recursively as $M.V_{ref*} = M.V_{ref} \cup M.SM.V_{ref*}$. Assuming that q is the current valuation of the variables in V_*, the set $V_\sharp$ is defined as $M.V_\sharp = M.V_* \cup \bigcup_{v \in M.V_{ref*}} q(v).V_g$. For every composite mode, the set $V_\sharp$ is extended with a local variable h which stores the currently active submode. In case no submode is active, h is valued ϵ. The state of the mode consists of a valuation of $V_\sharp$, denoted by various forms of q. The configuration state is captured by the valuation of the reference variables of the mode.

For any transition $(e, \alpha, x) \in M.T$ of a mode M, the action α is defined as a relation between the states of the variables. As described in the syntax, agents can be created or destroyed in α and as such also affect the system level. Respecting the hierarchy, however, the mode semantics cannot capture this directly. Therefore the relation defining the action is augmented with a list of agents created and destroyed in the action. Such a list will be denoted by various forms of L. The relation part of the action is a relation between $\mathcal{Q}_{V_{act} \setminus V_l}$ and $\mathcal{Q}_{V_{act}}$ if $e \in M.E$ and between $\mathcal{Q}_{V_{act}}$ and $\mathcal{Q}_{V_{act}}$ otherwise. All operations and assignments in α are executed sequentially, atomically, and instantaneously. An augmented pair $((q, q'), L) \in \alpha$ if and only if:

- q satisfies the guard γ attached to α.
- Assuming that α contains k operations, there is a sequence of pairs of a state and a list of created and destroyed agents $(q_1, L_1), (q_2, L_2), \ldots, (q_{k+1}, L_{k+1})$ such that $q_1 = q$, $L_1 = \langle \rangle$, and for every operation i, $1 \le i \le k$:
  - Unless specified otherwise, for every $v \in V_\sharp$, $q_{i+1}(v) = q_i(v)$.
  - If operation i is a create operation $(v_r, Init, S)$, then $L_{i+1} = L_i \frown \langle A_{new} \rangle$ with agent $A_{new} = (S.TM, S.V, Init)$ of structure S. Moreover $q_{i+1}(v_r) = A_{new}$ and $q_{i+1}[A_{new}.V] = Init$. Note that A_{new} is created instantly and can be used in operations of the remainder of the transition.

- If operation i is a destroy operation with argument v_d, then $L_{i+1} = L_i \frown \langle q_i(v_d) \rangle$. Moreover $q_{i+1}(v_d) = \epsilon$ and $q_{i+1}(v) = \epsilon$ for every reference variable $v \in V_\sharp$ with $q_i(v) = q_i(v_d)$.
- If operation i is neither a create nor a destroy operation, then $L_{i+1} = L_i$ and q_{i+1} is the result of the assignment operation performed in q_i.

- $q' = q_{k+1}$ and $L = L_{k+1}$.

The relations which capture the discrete steps of a mode M (R_D, R_e for $e \in M.E$, and R_x, for $x \in M.X$) are constructed from one or more transitions. As in Charon, the relations are constructed by sequentially aggregating the actions of the transitions, including the added lists of created and destroyed agents.

An atomic mode has a single internal step, which is the idling step. It is enabled if and only if the invariant of the mode is satisfied. Obviously, no agents are created or destroyed. So, for each state q such that $I(q)$, $((q,q), \langle \rangle) \in R_D$. Further an atomic mode can be entered and exited at any time. Since it does not have any entry or exit transitions, neither the state is changed nor agents are created or destroyed. That is, for all q, $((q,q), \langle \rangle) \in R_{de}$ and $((q,q), \langle \rangle) \in R_{dx}$.

For a composite mode M, the entry relations R_e and the exit relations R_x are constructed from the actions of entry, respectively exit transitions of the submodes of M. For each entry transition (e, α, e'), it holds that $((q,q'), L) \in R_e$ if for some q'', $((q,q''), L'') \in \alpha$, e' is an entry point of a submode M', $(q'', q', L') \in M'.R_{e'}$ and $L = L'' \frown L'$. For the default entry point, $((q,q), \langle \rangle) \in R_{de}$ whenever $q(h) \neq \epsilon$, which means that the execution of M has been previously interrupted by a group transition. None of the group transitions added in Charon contains a create or destroy operation and hence the create and destroy list is empty. When $q(h) = \epsilon$, a non-deterministic initialization occurs and thus $((q,q'), L) \in R_{de}$ whenever $((q,q'), L) \in R_e$ for some non-default entry point e. Similarly, for each exit transition (x', α, x) of a composite mode, $((q,q'), L) \in R_x$ if for some q'', $((q,q''), L'') \in M'.R_{x'}$, $((q'', q'), L') \in \alpha$ and $L = L'' \frown L'$. Also, M can be interrupted by a group transition at any moment during its execution and thus always has to be ready to exit through the default exit. Therefore, for every q such that $q(h) \neq \epsilon$, $((q,q), \langle \rangle) \in R_{dx}$.

Internal steps of a composite mode M are either internal steps of M changing the currently active submode or internal steps of the currently active submode of M. If a transition of the mode is involved in the step, then the source submode of the transition should be the active submode and allow an exit step that matches the transition, and also the target submode of the transition should allow a matching entry step. Similar to entry and exit steps, the create and destroy lists are constructed straightforwardly from the create and destroy lists of the transitions within the step. Consequently, $((q,q'), L) \in R_D$ if there exists a state q_0 such that $q_0[V] = q[V]$ and

- For an active submode $N \in M.SM$, it holds that $((q_0[N.V_\sharp], q'[N.V_\sharp]), L) \in N.R_D$ and $q_0[V_\sharp \backslash N.V_\sharp] = q'[V_\sharp \backslash N.V_\sharp]$, or
- The following four conditions hold:
  - There exists an exit point x of the active submode N such that for some q_1 and L_1, $((q_0[N.V_\sharp], q_1[N.V_\sharp]), L_1) \in N.R_x$.

- There exists an entry point e of a submode N' such that for some q_2 and L_2, $((q_2[N'.V_*], q'[N'.V_*]), L_2) \in N'.R_e$.
- There exists a transition $(x, \alpha, e) \in M.T$ such that for some L_3, $((q_1, q_2), L_3) \in \alpha$.
- $L = (L_1 \frown L_3) \frown L_2$.

Similar to the Charon mode semantics, the continuous steps are captured by the relation R_C. The relation $R_C \subseteq \mathcal{Q}_{V_+} \times \mathcal{F}_{V_+}$ gives for every state q of M, the set of flows from q. R_C is obtained from the constraints of a mode and relation $N.R_C$ of its active sub-mode. Given a state Q of a mode M, $(q, f) \in R_C$ if and only if the following three conditions hold:

- The flow f is permitted by M, i.e., f satisfies all constraints in $M.Cons$.
- $(q[N.V_\sharp], f[N.V_\sharp]) \in N.R_C$.
- For each variable x, $q(x) = f(0)(x)$ unless M has an algebraic constraint A_x.

To be able to define the semantics for a mode, all reference variables used in a mode must be initialized, i.e., be unequal to ϵ, at all times at which the mode is active. A reference variable is used if it either appears in a destroy operation or a global variable of the referenced agent appears in the constraints or transitions. Consequently, every mode should have an invariant for each used reference variable, which states that the reference variable is unequal to ϵ. Figure 5 shows an example of a mode $M1$ with one reference variable ref_1 complying to this requirement.
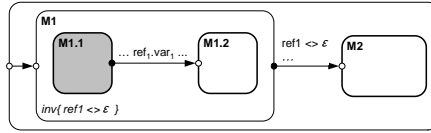

Fig. 5. Example of an additionally required invariant for a used reference variable

Definition 5 (Mode Operational semantics). *The operational semantics of the mode M, $OS(M)$ is defined to be a six tuple consisting of its control points, its variables and the discrete and continuous relations: $OS(M) = (M.E \cup M.X, M.V_\sharp, M.R_C, M.R_D, \{M.R_e | e \in M.E\}, \{M.R_x | x \in M.X\})$.*

Agent Operational Semantics. Assume that q is the current valuation of the variables in V. Denote the set of all variables as well as all global variables of referenced agents as $V_\sharp$, formally $V_\sharp = V \bigcup_{v \in V_r} q(v).V_g$. The state of the agent consists of a valuation of $V_\sharp$, denoted by various forms of q.

To improve the clarity of the semantics of the system level, we lift the discrete and continuous step relations defined in the mode semantics to the level of

the agent semantics. The discrete and continuous relations RA_{init}, RA_D, and RA_C are constructed from the relations R_{init}, R_D, and R_C respectively of the top-level modes of the agent[3]. For $o \in \{init, D\}$, $((q_1, q_2), L) \in RA_o$ if and only if there is an $M \in TM$ such that $((q_1[M.V_\sharp], q_2[M.V_\sharp]), L) \in M.R_o$. For the continuous steps, $(q_1, f) \in RA_C$ if and if only for every mode $M \in TM$, $(q_1[M.V], f[M.V]) \in M.R_C$.

Definition 6 (Agent Operational semantics). *The operational semantics of the agent A, $OS(A)$ is defined to be a five tuple consisting of the control points of its top-level modes, its variables, the continuous step relation, and the discrete step relations: $OS(A) = (A.TM.E \cup A.TM.X, A.V_\sharp, A.RA_C, A.RA_D, A.RA_{init})$*

System Operational Semantics. The state of a system consists of a two tuple $(q, \mathcal{A})$. The first element of this tuple is a valuation of all variables of all agents in $\mathcal{A}$. The set of variables of $V_\sharp$ of the system is defined as $V_\sharp = \mathcal{A}.V$. The second element of the tuple is the set of all agents that currently exist in the system.

Definition 7 (System operational semantics). *The operational semantics of a system Sys, $SO(Sys)$ consist of the set of structures and the set of agents of the system: $SO(Sys) = (\mathcal{S}, \mathcal{A})$.*

The operational semantics define a transition system $\mathcal{RS}$ over the states of the system Sys. A transition of the system $\mathcal{RS}$ representing a continuous step is denoted by $(q_1, \mathcal{A}) \overset{(f,t)}{\to} (q_2, \mathcal{A})$ if for all $A \in \mathcal{A}$, $(q_1[A.V], f[A.V]) \in A.RA_C$ with f defined on the interval $[0, t]$ and $f(t) = q_2$. Agents created and destroyed in a discrete step will be added, respectively deleted from the system subsequent to the discrete step. Each destroyed agent will be deleted from the system in a system update delete step. Each created agent will be added in a system update add step which is immediately followed by the discrete initialization steps of the top-level modes of the created agent. System update steps and discrete initialization steps of agents created or destroyed during a discrete initialization step are handled in a depth-first approach. A discrete step and its aftermath are thus best described recursively. For this purpose we introduce the recursive function Γ with as arguments a state $(q, \mathcal{A})$ and a list L of created and destroyed agents. The function $\Gamma((q, \mathcal{A}), L)$ returns all possible parts of the transition system dealing with recursively adding and initializing the agents created and deleting the agents destroyed during the discrete step.

Assuming the function Γ, a transition of the system representing a discrete step and its aftermath is denoted by $(q_1, \mathcal{A}) \overset{o}{\to} \Gamma((q_2, \mathcal{A}), L)$ if there is an $A \in \mathcal{A}$ such that $((q_1[A.V], q_2[A.V]), L) \in A.RA_o$ for $o \in \{init, D\}$. In a discrete step only one top-level mode of one agent takes a discrete step.

Both the system update add and delete step only occur in the parts of the transition system defined by Γ and are defined in the context of Γ. The function

---

[3] A Charon top-level mode only has a single entry point *init* and no exit points.

Γ is defined based on the pattern of the list of created and destroyed agents argument. In case the list is empty, no agents have been created or destroyed during the discrete step: $\Gamma((q, \mathcal{A})), \langle\rangle) = (q, \mathcal{A})$. In case the first element of the list is a destroyed agent A_d, first a system update delete step is taken: $\Gamma((q_1, \mathcal{A})), \langle A_d \rangle \frown L) = (q_1, \mathcal{A}) \stackrel{u_d}{\rightarrow} \Gamma((q_2, \mathcal{A}\backslash\{A_d\}), L)$ if

- $q_2[V_\sharp \backslash \mathcal{A}.V_r] = q_1[V_\sharp \backslash \mathcal{A}.V_r]$.
- For all $v \in \mathcal{A}.V_r$, if $q_1(v) = A_d$ then $q_2(v) = \epsilon$ otherwise $q_2(v) = q_1(v)$. That is, all references to the destroyed agent are removed.

In case the first element of the list is a created agent A_c:

$$\Gamma((q, \mathcal{A}), \langle A_c \rangle \frown L) = (q, \mathcal{A}) \stackrel{u_a}{\rightarrow} (q_0, \mathcal{A}_0) \stackrel{init}{\rightarrow}$$

$$\Gamma((q_1, \mathcal{A}_1), L_1) \stackrel{init}{\rightarrow} \ldots \stackrel{init}{\rightarrow} \Gamma((q_{k-1}, \mathcal{A}_{k-1}), L_{k-1}) \stackrel{init}{\rightarrow} \Gamma((q_k, \mathcal{A}_k), L_k \frown L)$$

if

- Agent A_c has k top-level modes, i.e., $|A_c.TM| = k$.
- $q_0[A.c\backslash(A_c.V_r \cap A_c.V_g)] \in A_c.I[A.c\backslash(A_c.V_r \cap A_c.V_g)]$. Note that the valuation of global reference variables of A_c already might have been changed by the remainder of the discrete step in which the agent has been created.
- The created agent is added to the set of agents in the system update step, i.e., $\mathcal{A}_0 = \mathcal{A}_1 = \mathcal{A} \cup \{A_c\}$.
- There is an $M \in A_c.TM$ such that $((q_0[A_c.V], q_1[A_c.V]), L_1) \in M.R_{init}$. That is, the first discrete step initializes one of the top-level modes of the added agent.
- For every $2 \leq i \leq k$:
  - Denote the last state of the transition system part defined by $\Gamma((q_i, \mathcal{A}_i), L_i)$ as $(q'_i, \mathcal{A}'_i)$.
  - There is an $M' \in A_c.TM\backslash M$ such that $((q'_{i-1}, q_i), L_i) \in M'.R_{init}$. That is, the remaining top-level modes of the added agent are initialized.
  - $\mathcal{A}_i = \mathcal{A}'_{i-1}$.

An execution of a system $Sys = (\mathcal{S}, \mathcal{A})$ is a path through the transition graph of $\mathcal{RS}$ and starts with:

$$(q_0, \mathcal{A}_0) \stackrel{init}{\rightarrow} \Gamma((q_1, \mathcal{A}_1), L_1) \stackrel{init}{\rightarrow} \Gamma((q_2, \mathcal{A}_2), L_2) \stackrel{init}{\rightarrow} \ldots \stackrel{init}{\rightarrow} \Gamma((q_k, \mathcal{A}_k), L_k)$$

such that if we define $(q'_i, \mathcal{A}'_i)$ to denote the last state of a transition system part defined by $\Gamma((q_i, \mathcal{A}_i), L_i)$:

- $\mathcal{A}_0 = \mathcal{A}$ and for all $A \in \mathcal{A}$ it holds that $q_0[A] \in A.I$.
- The number k equals the total number of top-level modes initially in the system, i.e., $k = \sum_{A \in \mathcal{A}} |A.TM|$.
- Each of the k top-level modes initially in the system takes one of the k explicitly described discrete initialization steps.
- For any i, $2 \leq i \leq k$ it holds that $q_i = q'_{i-1}$ and $\mathcal{A}_i = \mathcal{A}'_{i-1}$.

From that point on the execution continues as follows:

$$\stackrel{(f_1, t_1)}{\rightarrow} (q_{k+1}, \mathcal{A}_{k+1}) \stackrel{o}{\rightarrow} \Gamma((q_{k+2}, \mathcal{A}_{k+2}), L_{k+2}) \stackrel{f_2, t_2}{\rightarrow} \ldots$$

such that for any $i > 0$, i odd, it holds that $\mathcal{A}_{k+i} = \mathcal{A}'_{k+i-1}$.

5   Example Revisited

We discuss a part of a trace of the air-traffic control example system of Section 3:

$$\ldots\ (q_0, \mathcal{A}) \overset{D}{\to} (q_1, \mathcal{A}) \overset{u_a}{\to} (q_1, \mathcal{A}') \overset{init}{\to} (q_2, \mathcal{A}') \overset{(f_1, 0)}{\to} (q_2, \mathcal{A}') \overset{D}{\to}$$

$$(q_3, \mathcal{A}') \overset{(f_2, t_2)}{\to} (q_4, \mathcal{A}') \overset{D}{\to} (q_5, \mathcal{A}') \overset{u_d}{\to} (q_6, \mathcal{A}) \overset{(f_3, 0)}{\to} (q_6, \mathcal{A}) \overset{D}{\to} (q_7, \mathcal{A}) \overset{(f_3, 0)}{\to} \ldots$$

We consider the system at a stage with three agents: a ground control agent, a center agent, and an airplane agent, i.e., $\mathcal{A} = \{gc, ctr, a\}$. Assuming the airplane is approaching the no-fly zone, a discrete step in the gc agent occurs. In this step, gc creates a new controller c to guide a which results in a system update add step and $\mathcal{A}' = \{gc, ctr, a, c\}$. Note that the valuation q_1 does not change in this step. As described in the semantics, the add step is followed by the discrete initialization step of c. In this initialization step a link from a to c is created, i.e., $q_2(a.con) = c$. The continuous step with flow f_1 has a duration of 0 because the invariant of the *Free flight* mode of a evaluates to false now. The next discrete step is then a mode switch in a to the *Ground control guided* mode. After some time t_2 the airplane has been navigated successfully around the no-fly zone and the controller c destroys itself in a discrete step. This leads to the system update delete step in which the link from a to c is removed ($q_6(a.con) = \epsilon$) and c is removed from the system. Because the invariant in the active mode of a has become false again, the next continuous step has a duration of 0. In the following discrete step, a is forced back into the *Free flight* mode.

6   Conclusion and Future Work

We have presented an extension for reconfigurability to Charon, the hierarchical modular language for hybrid systems. The presented extension is a semi-conservative extension of Charon. i.e., an embedding of a Charon model to an R-Charon model exists [18]. The language extension is designed to support physical as well as communication-wise reconfiguration as encountered in large-scale transportation systems and in modular reconfigurable robots. Applicability of the reconfiguration notion inspired by SHIFT has already been shown [19].

The compositionality results of R-Charon modes can be taken over and extended straightforwardly from the mode compositionality results in [3]. A logical next step is to come up with a sound notion of agent compositionality and to prove that it holds for R-Charon agents. Other relevant work includes extending the Charon toolkit to support the presented reconfiguration concept, applying R-Charon to real modular robot models and use the models for analysis, exploring explicit agent hierarchy and reconfiguration between sub-agents within an agent, and adding a location model as a first class language element.

References

1. Fromherz, M.P.J., Crawford, L.S., Hindi, H.A.: Coordinated control for highly reconfigurable systems. In: HSCC '05: Proceedings of the 8th International Workshop on Hybrid Systems. Volume 3414 of LNCS. Springer-Verlag (2005) 1 – 24
2. Alur, R., Grosu, R., Hur, Y., Kumar, V., Lee, I.: Modular specification of hybrid systems in Charon. In: HSCC '00: Proceedings of the 3rd International Workshop on Hybrid Systems. Volume 1790 of LNCS. (2000) 6 – 19
3. Alur, R., Grosu, R., Lee, I., Sokolsky, O.: Compositional modeling and refinement for hierarchical hybrid systems. Journal of Logic and Algebraic Programming (**To appear**)
4. Cuijpers, P.J.L., Reniers, M.A.: Hybrid process algebra. Journal of Logic and Algebraic Programming **62**(2) (2005) 191 – 245
5. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid I/O automata. Information and Computation **185**(1) (2003) 105 – 157
6. Rounds, W.C., Song, H.: The ϕ-calculus: A language for distributed control of reconfigurable embedded systems. In: HSCC '03: Proceedings of the 6th International Workshop on Hybrid Systems. Volume 2623 of LNCS. (2003) 435 – 449
7. Varaiya, P.: Smart cars on smart roads: problems of control. IEEE Transactions on Automatic Control **38**(2) (1993) 195 – 207
8. Zelinski, S., Koo, T.J., Sastry, S.: Hybrid system design for formations of autonomous vehicles. In: 42nd IEEE Conference on Decision and Control. Volume 1. (2003) 1 – 6
9. Perry, T.S.: In search of the future of air traffic control. IEEE Spectrum **34**(8) (1997) 18 – 35
10. Bishop, J., Burden, S., Klavins, E., Kreisberg, R., Malone, W., Napp, N., Nguyen, T.: Self-organizing programmable parts. In: International Conference on Intelligent Robots and Systems. (2005)
11. Yim, M., Zhang, Z., Duff, D.: Modular robots. IEEE Spectrum **39**(2) (2002) 30 – 34
12. Østergaard, E.H.: Distributed Control of the ATRON Self-Reconfigurable Robot. PhD thesis, Maersk McKinney Moller Institute for Production Technology, University of Southern Denmark (2004)
13. Deshpande, A., Göllü, A., Semenzato, L.: The SHIFT programming language and run-time system for dynamic networks of hybrid systems. IEEE Transactions on Automatic Control **43**(4) (1998) 584 – 587
14. Milner, R.: Communicating and Mobile Systems: the π-calculus. Cambridge University Press (1999)
15. Aldini, A., Bernardo, M.: On the usability of process algebra: An architectural view. Theoretical Computer Science **335**(2-3) (2005) 281 – 329
16. Attie, P.C., Lynch, N.A.: Dynamic input/output automata: a formal model for dynamic systems. In: CONCUR 2001: 12th International Conference on Concurrency Theory, Aaalborg, Denmark. (2001)
17. Teich, J., Koster, M.: (Self-)reconfigurable finite state machines: Theory and implementation. In: Proceedings of the conference on Design, automation and test in Europe. (2002) 559 – 568
18. Kratz, F.: A modeling language for reconfigurable distributed hybrid systems. Master's thesis, Technische Universiteit Eindhoven (2005)
19. Antoniotti, M., Göllü, A.: SHIFT and SmartAHS: A language for hybrid systems engineering, modeling, and simulation. In: Proceedings of the USENIX Conference of Domain Specific Languages. (1997)