Best Practice in Robotics(BRICS)
Grant Agreement Number: 231940
01.03.2009 - 28.02.2013

Deliverable D4.2: Statecharts and IPC policy improvements to MDE standards

Markus Klotzbuecher, Herman Bruyninckx

Deliverable: *D*4.2

Consortium	**Document Info**	
KUKA Roboter GmbH	Deliverable	D4.2
GPS GmbH	Dissemination	Restricted/Public
BRSU	Status	Final
KUL	Lead Contractor for Deliverable	Katholieke Universiteit Leuven
Fraunhofer IPA	Due Date of Deliverable	March 1, 2010
UTwente	Actual Submission Date	June 8, 2011
UBergamo	Version	0.2
BLUEBOTICS	Pages	10

Ver	Date	Author	Description
0.1	February 28, 2010	Markus Klotzbücher, Herman Bruyninckx	Initial version
0.2	June 8, 2010	Markus Klotzbücher, Herman Bruyninckx	Addressed internal reviewers comments

Contents

1 Executive Summary

Finite state machine based formalisms such as the OMG UML2 State Machines or Harel Statecharts are widley used for modeling the behavior of complex systems. However applying these formalisms for modeling behavior in component based, robotic systems reveals several shortcomings. The subject of this work has been to analyse the most commonly used finite state machine based modeling standards with respect to their applicability to the robotics domain. Based on these insights we have derived a Statechart model which is targeted to the domain of *Coordination* of complex robotic systems and yet minimalistic in terms of provided model elements.

To complement the semantic analysis a reference implementation has been developed, which has already been successfully applied at several occasions, including the *two arm KUKA Lightweight Robot Arm setup* at the BRICS research camp in Malaga.

The work carried out is only outlined in this deliverable; a full description on the topic of semantic analysis to Statecharts can be found found in the following publication [6] (draft status), which we intend to submit to the JOSER Journal of Software Engineering. The real-time aspects of the software implementation are described in detail in this publication [7], which was presented at the International workshop on Dynamic Languages for Robotic and Sensors.

Improvements to IPC standards are discussed in the context of the *BRICS component model*. The major insight gained here is that the aspect of *Communication* must not be included in the component model, but instead needs to be factored out into an individual model.

2 Overview: the five C's

The current work is carried under the hypothesis that the following aspects (called the 5C's) must be separated in order to construct robust and reusable systems. Figure 2.1

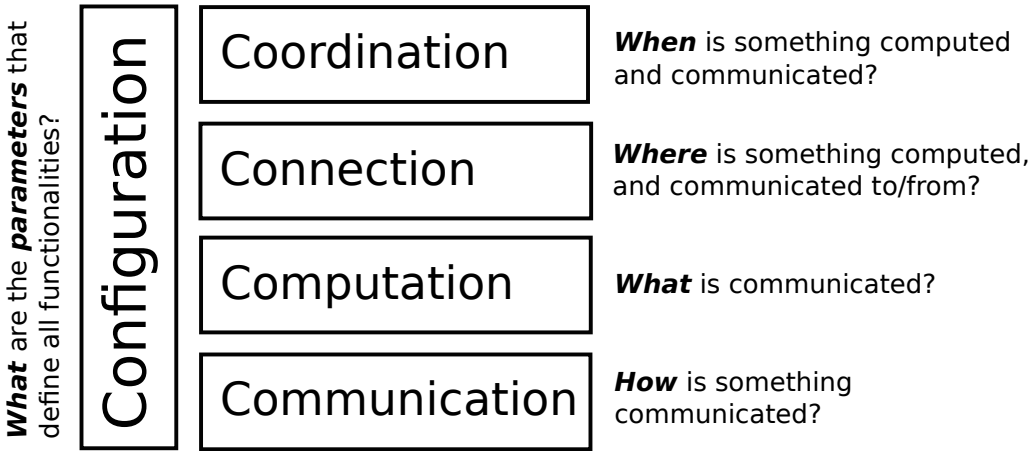

Figure 2.1: The five C's

The aspect Communication defines in which way parts of the system can interact with each other such as by anonymous message passing or request-reply interaction. Computation defines the elementary functionality of which the system is composed. Configuration defines defines parameters of all other aspects. For instance this could be which computations form a system and their parameters. Connection defines how these computations are interconnected. At last Coordination manages the computations and communication such that the behavior of the system emerges as intended.

3 Statechart improvements to MDE standards

Overview

Finite state machine based models and implementations have been in use for decades [3], [4], [2], [10] [9]. As a consequence many different variants exist which significantly differ in their semantics, target domain and provided feature set. In order to determine a statechart model suitable for the domain of *Coordination* in robotics, existing coordination models were analyzed in order to identify a minimal subset for this purpose. Summarized, the following criteria were taken into account:

- Suitability for coordination of complex, multi-robot systems.
- *Composability*: it must be possible to create complex models by combining simple ones.
- *Compositional robustness*: changes or errors in individual parts shall be *localized* as much as possible effect the whole system as little as possible.
- *Minimality*: the model shall be as small and simple as possible.
- *Relation to UML*: as UML is widely used as a modeling language, the graphical notation and semantics shall be followed as much as possible in order to facilitate adoption by users already familiar with this standard.

Based on these criteria the rFSM (reduced finite state machine) formalism was defined, which is described in the next section.

The reduced Finite State Machine model

This is a brief overview of the rFSM (reduce Finite State Machines) formalism. It is a minimalistic model, yet sufficiently rich to serve as coordination component for most of the robot control software. The model has a **structural** part (describing states and their transition interconnections), and a **behavioral** part (describing the activities of the robot system when in a particular state, or performing a particular transition).

Structural Model

The rFSM state machine model is a minimal subset of UML2 and Harel Statecharts. It consists of the following four, main model elements:

- Simple state
- Composite state
- Transition
- Connector

In addition two virtual model elements are introduced in order to simplify descriptions about different types of elements:

- *States* are either of simple state or composite state type.
- *Nodes* are either States or Connectors.

Figure 3.1 shows the complete model as an Ecore diagram.

A composite state is a state which can contain either other composite states or simple states. At the top-level any rFSM model is always contained in a top-level composite state. This way a state machine can immediately be composed by inserting it into a new composite state.

In contrast to composite states *simple states* can not contain any other states; they are leaves in the state machine tree. (This *tree* is not to be confused with the state machine *graph*, in that the tree represents a hierarchy of decomposition, and not a map of the transitions that can take place between states.) Transitions connect Nodes in a directed fashion and carry a list of events

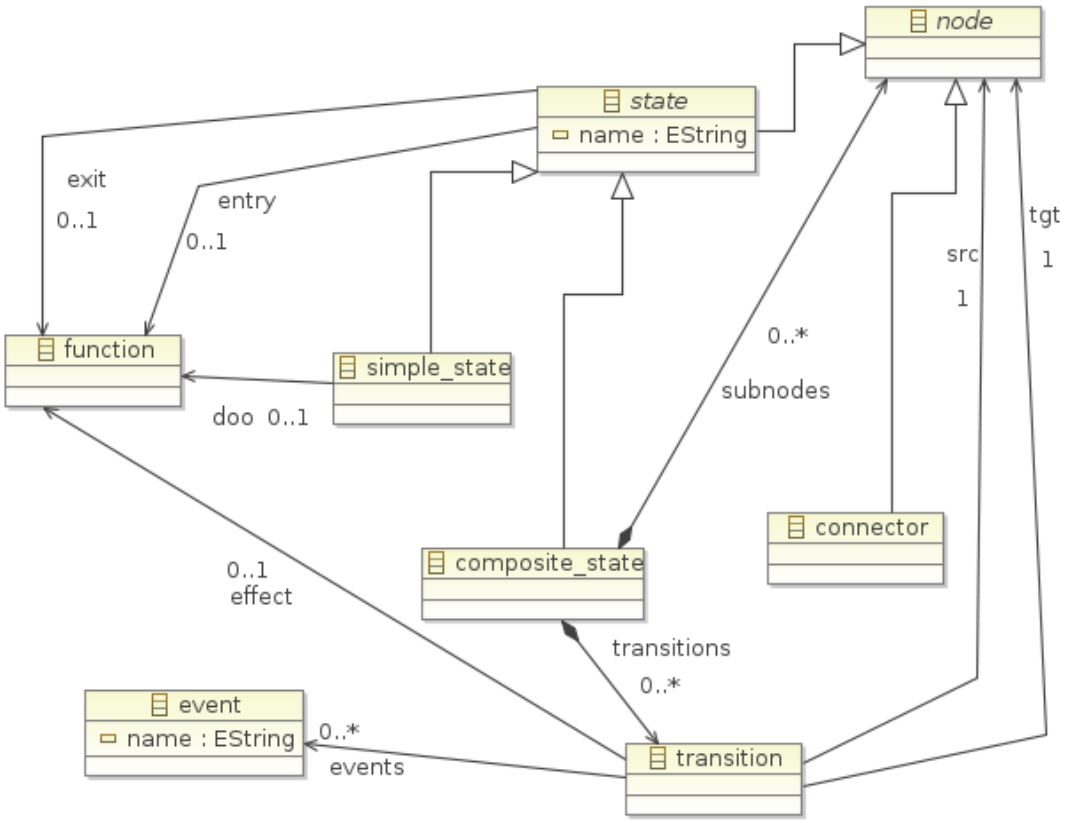

Figure 3.1: rFSM Ecore model

which will trigger the transition. Transitions are owned by a composite state and not (as often assumed) by the state from which they originate.

Connectors can be used to build complex transitions by interconnecting several elementary ones. This model element unifies the four very similar UML model elements junction, initial, entry- and exit pseudostates.

While connectors can join together multiple transitions it is required that any complex transition must always start and end on a State.

There exists one connector with special semantics: the initial connector. When a transition which ends on the boundary of a composite state is executed, the execution will continue with the transition emanating from the initial connector. Static checks assure that each composite state which is the target of a transition also contains a initial connector.

Both States and transitions can be associated with programs. States may have entry and/or exit programs which are executed when the state is entered or left respectively. Simple states may in addition define a `do` program which is executed while the state is active. Transitions may define a `guard` condition and an `effect` program. The guard condition is evaluated when a transition is considered to be executed and will inhibit the transition if `false`. The `effect` function is only executed once the transition is taken.

Behavioral model

After having introduced the building blocks which form the rFSM model, we now describe the run-time behavior.

In classical finite state automatons only one state may be active at a time. In contrast the Statecharts formalism allows multiple states to be active. The constraints under which this is allowed are:

- for any active state its parent state must be active too
- in a composite state only one child state may be active at a time

A state-machine is executed for the first time by executing the transition starting from the initial connector which will result in the target state of this transition to be entered.

The elementary way to advance the state machine is to invoke its `step` procedure. The step procedure will take *all* events which accumulated since the last step and attempt to find an enabled transition. This process starts top down, starting from the root composite state down to the active leaf simple state. As soon as a transition is found the searching is finished and the transition is executed.

This approach of identifying the next transition has the advantage that it assigns explicit priorities (`structural priorities`) to transitions (higher to less deeply nested transitions) which are visible in the graphical representation. Given a set of events and the current active states of the state graph it is immediately visible which transition will be taken. This follows the approach of the STATEMATE semantics [4]. Furthermore structural priority largely avoids conflicts among emanating transitions, leaving only the possibility of conflicts for transitions leaving a single state. These can be eliminated either by additional guard conditions or by means of explicitly defining their priorities (priority numbers).

Parallel states

The rFSM model does not include a parallel state model element. This has several reasons. Most importantly, rFSM is designed for modeling Coordination. In contrast to Computation, coordinative actions generally consist of issuing commands to lower level Computation components and do not require intensive computations. Secondly, to define the semantics of parallel states requires introducing many fundamental assumptions such as how parallelism is achieved (threads, processes, etc.), how priorities are assigned between parallel regions or even fundamental issues such at which level parallelism shall be introduced (e.g. only parallel do programs or parallel state entry). Thus it is preferable to not introduce these assumptions into the core semantics if not strictly necessary.

Of more importance than parallel states (and a practical replacement for it) are distributed state machines which, for instance, allow to express relationships between multiple robots. Each of the distributed state machines is an example of the above-mentioned centralised, single-threaded, state machine. Additionally, by providing a mechanism to distribute state machines in general the special case of parallel states (distribution at thread level) is covered.

Model transformation between rFSM and UML

Given that rFSM is structurally a subset of UML 2, it is straightforward to define a model transformation from UML to rFSM, thereby making it possible to reuse the large set of available UML modeling tools to define rFSM models. Of course such transformations require that either the UML model elements not available in rFSM can be replaced by composition of rFSM primitives or are not used during modeling. Violations of this constraint can easily be detected during transformation.

Examples

BRICS research camp dual arm handover

The following application was prepared as a basic starting point for researchers working on the BRICS research on mobile manipulation. The overall setup is shown in figure 3.2.

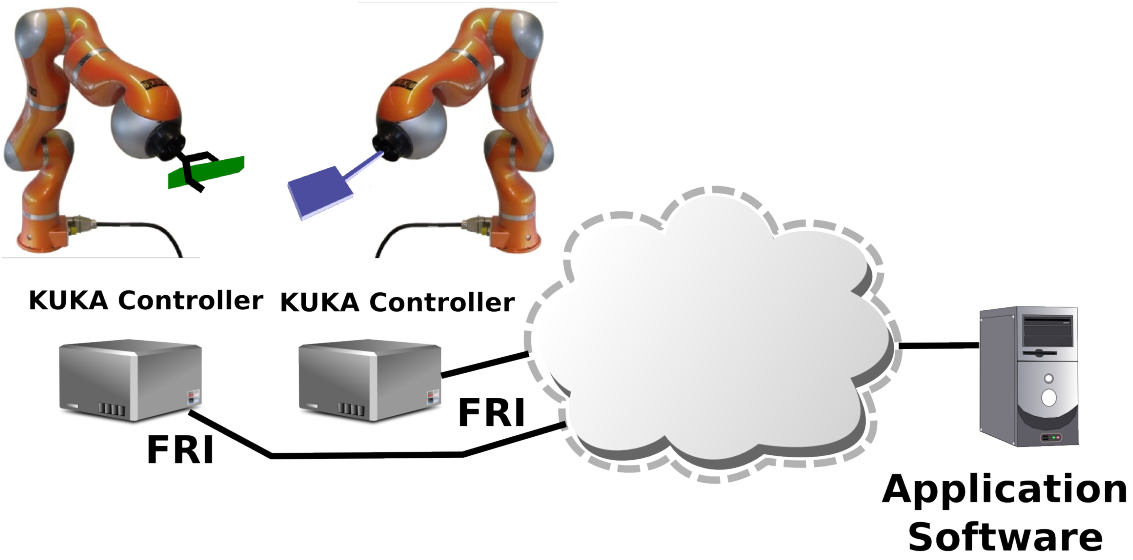

Figure 3.2: Dual arm handover setup

The application involves two KUKA light weight robot arms (LWR) which interact with each other by handing over an object from one arm to the other. Besides the coordination necessary for the application itself, further coordination is necessary for interacting with the KUKA Fast research interface (FRI) [8]. The FRI interface provides low-level real-time access to the LWR and can reside in two basic states: in *monitor mode* the robot state can be read; only in *command mode* the robot can be additionally controlled. The FRI state may switch from command mode to monitor mode if the communication quality is insufficient and thus stop executing commands. Consequently for the application to work both robots must be in command mode.

Figure 3.3 shows the rFSM state chart which models the constraints between application and FRI states. The `handover_demo` state containts the application specific states of the robot arms while handing over the object. If a controller enters the FRI monitor state due to bad communication quality, a transition to the `waiting` state is taken in which both robots are paused. This transition prevents that one robot continues the handover while the other is unoperational. After the both FRI Controllers are back in command mode, the application can continue.

This example shows the suitability of statecharts for expressing dependencies among different coordination. The fact that the application may only execute while the FRI Controller is in command mode is expressed by the two toplevels states independently of the application. This separation would not be possible if this coordination were programmed.

Ball tracking example

An example of a rFSM model is given by the component assembly shown in figure 3.4. A ball swinging on a string is to be followed by a robot arm. The 2D ball positions extracted from two camera images by BallExtractor components are passed to the estimation component. The estimated value is then sent to the RobotController which actuates the robot arm. Now the

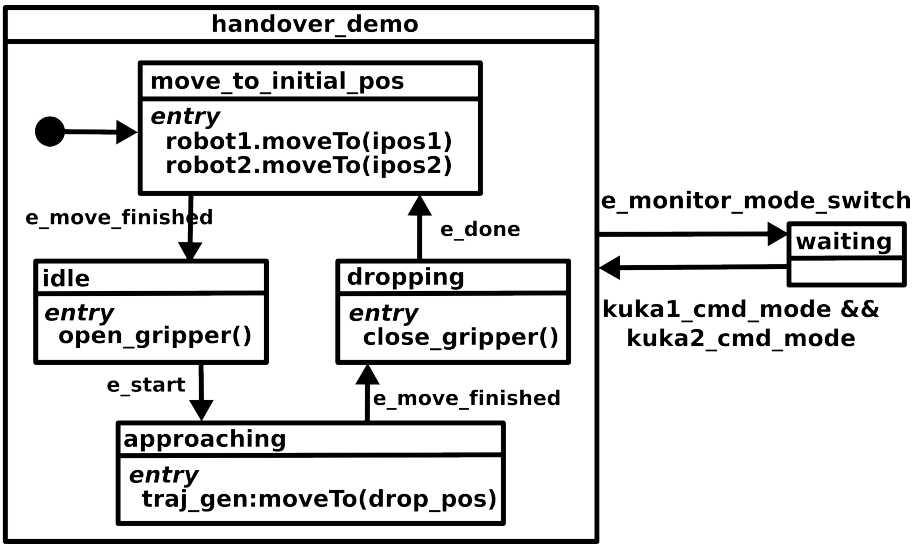

Figure 3.3: Extended coordination taking FRI states into account

situation is possible that the ball swings out of the observed camera range. In this case the desired behavior is that the robot arm stops at the last estimated ball position. However different estimation models will show different behaviors; for instance a constant velocity model will predict the ball motion to continue with the last estimated velocity while a constant position model will continue to predict the last observed position.

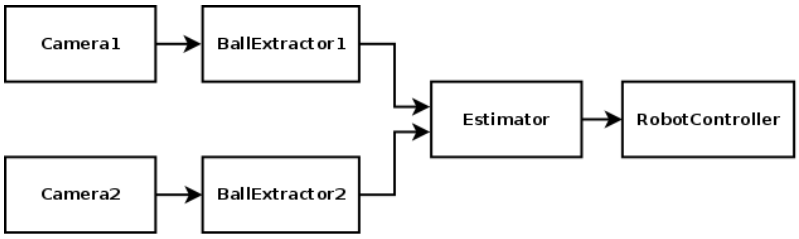

Figure 3.4: Example component layout

A naive solution to this problem would be to add a feature to the estimator to stop the robot controller once the ball has left the camera range. This solution however will severely limit the reusability of this component and make it impossible to replace it with different estimation components which do not make this application dependent assumption. A better solution is to introduce a *Coordination component* which encapsulates this policy. The state machine for such a component is shown in figure 3.5.

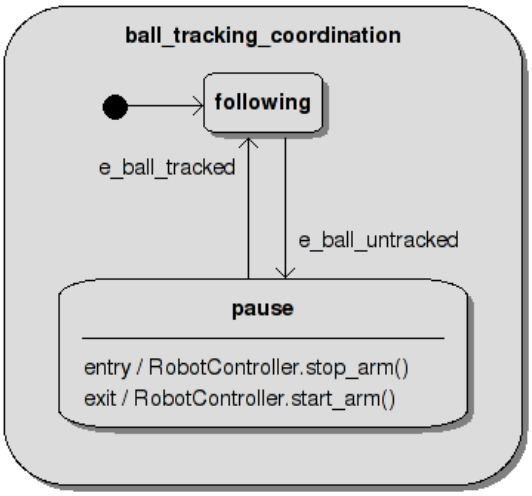

Figure 3.5: rFSM example

Instead of making assumptions about the component layout the estimation component raises an event when the ball is not being tracked anymore. The Coordination component then reacts to this event and transitions to the `pause` state in which the robot is stopped. When the ball enters the camera range and the estimator begins tracking the ball again, a second event is raised to transition back to the `following` state and restarting the robot controller in the exit program of the pause state.

rFSM Software Implementation

We have developed a lightweight and real-time implementation of reduced rFSM Statecharts using the Lua [5] scripting language. The approach for achieving real-time safe execution of rFSM statechart models is described in detail in this publication [7] which was presented at the International workshop on Dynamic Languages for Robotic and Sensors in Darmstadt.

4 IPC policy improvements

The underlying mechanisms of Inter-Process communication are most of the time intentionally invisible to the robot system designer. Instead it is left up to the middleware to determine which option is best suited for the specific case. For instance when large amounts of data are shared between local computations on a single host, the middleware will select a zero-copy, shared memory based IPC mechanism; likewise a POSIX message queue might be chosen in the case that lowest possible communication latencies are required.

While this transparency is conveniant during early system development, later on it often becomes necessary to modify the communication properties in order to optimize at system level. Achieving this at software framework level is straightforward; for instance OROCOS RTT [1] framework allows to define communication properties by means of the `ConnPolicy` type. A more difficult question is how to represent such information at the modeling level: on the one hand it is necessary include modeling support for such properties, as they pertain to the system specification. One the other hand this implies "promoting" low-level communication attributes, many which will not be available for all target frameworks, to the platform independent layer. The solution to this dilemma, which became apparent during the work of the Component Model Task Force is to express such communication specific problems using a separate Communication model. And in this Communication model, the behaviour of the communication can be treated by a Coordination FSM. This fact has become obvious as soon as we realised in the BRICS project that it was necessary to promote the Communication to become first-class components in the total component model.

Bibliography

[1] Herman Bruyninckx. Open Robot COntrol Software. http://www.orocos.org/, 2001. Last visited November 2010.

[2] Object Management Group. Uml 2.0 superstructure, ver. 2.1.2, formal/07-11-02, 2007.

[3] David Harel. State charts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[4] David Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Trans. on Software Engineering Methodolody*, 5(4):293–333, 1996.

[5] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Lua—an extensible extension language. *Softw. Pract. Exper.*, 26(6):635–652, 1996.

[6] Markus Klotzbuecher and Herman Bruyninckx. A minimal variant of UML state machines for modeling coordination in complex robotic systems (draft status), 2011.

[7] Markus Klotzbuecher, Peter Soetens, and Herman Bruyninckx. OROCOS RTT-Lua: an Execution Environment for building Real-time Robotic Domain Specific Languages. In *International Workshop on Dynamic languages for RObotic and Sensors*, 2010.

[8] Günter Schreiber, Andreas Stemmer, and Rainer Bischoff. The Fast Research Interface for the KUKA Lightweight Robot. In IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications – How to Modify and Enhance Commercial Controllers (ICRA 2010), May 2010.

[9] W3C. State chart xml (scxml): State machine notation for control abstraction. W3C Working Draft, 2010. http://www.w3.org/TR/scxml/.

[10] The Math Works. Stateflow, 2011. http://www.mathworks.com/products/stateflow/.