Specification and Verification of Complex Robotics Tasks

Rune M. Jensen
Department of Information Technology
Technical University of Denmark
Lyngby, Denmark

August 10, 1998

Abstract

This paper applies duration calculus to the specification and verification of a complex robotics task: Fingers grasping an object. We present a model of the relevant features of the mechanical design and provide a specification for sensors, actuators and a controller. Requirements are then specified in an assumption commitment style, and it is checked through calculation that the design satisfies the requirements.

1 Introduction

Developement of complex real-time robotics systems is a challenging engineering task. First, the development of system components is often divided among several subprojects using different design methodologies, which makes the final behaviour of the integrated robot system hard to predict and correct [ORS96]. Second, traditional design methods based on separate mathematical models of each electrical and mechanical component and statistical models of the timing behaviour of the controller programs has led to systems with chaotic behaviour, because performance is very sensitive to initial conditions [SF97].

To overcome these problems developers in robotics should consider integration of the components during the design. Moreover when specifying system requirements a coherent and mathematically well-founded formalism integrating the continuous dynamic properties with the discrete controller properties should be used to apply mathematical reasoning to determine whether a system design conforms to specific requirements.

Because robotic systems in contrast to other real-time control systems usually exhibit asynchronous behaviour, they cannot be modeled by states that change at fixed time steps [SF97]. Hence, a formalism applicable to systems with varying time transitions is required. Duration Calculus (DC) [ZHR91, HC97] is a temporal logic possessing these properties. Further, DC makes it possible to formalize properties of real-time hybrid systems with interacting continuous and discrete states using predicate logic and mathematical analysis.

In this paper we present a design approach suitable for robotic systems based on the *Requirements Language* (RL) defined by the Provably Correct Systems project (**ProCoS**) [HHF$^+$94]. RL embeds Duration Calculus in Z-schemas [Spi87, Rav95]. The Z-schemas introduce a module concept which enables structured declarations corresponding to the specifications of systems and subsystems [ORS96]. There are two advantages of this. First, it makes the specification easier to share between several developers, and second, it makes verification less complex by clustering quantities with associated formulas that only speak about these quantities [ORS96].

The design approach documents the system development through three phases: System modelling, requirements specification and behaviour verification. We use a visual grasping task presented in [SF97] as an example and verify a selection of sensors and actuators, with a desired program architecture with respect to system requirements and model.

A similar design approach has been used in formal specification of a gasburner [HHF$^+$94, Rav95], a railroad crossing [ORS96] and a digital controller of a hydraulic arm manipulator [RRH$^+$95]. Compared to this work we introduce a more exact specification of sensors and actuators and a more explicit system architecture. Also [SF97] uses a similar design approach, but here the specification of physical components is stated as controller phase requirements, which makes the design a program design rather than a system design.

The paper is organized as follows: Section 2 briefly introduces RL. In Section 3 a hierarchical model of the visual grasping robot is developed, which leads to specification of requirements in an assumption commitment style in Section 4. In Section 5 the requirements are verified by means of formal mathematical reasoning on the design and system assumptions. Finally in Section 6 we draw conclusions.

2 Introduction to *Requirement Language*

The main building blocks of RL are DC formulas embedded in Z-schemas. DC is a dense time temporal logic on time intervals and has evolved from Moszkowskis's (discrete time) interval logic (ITL) [Mos85]. A complete definition of DC and a proof system for DC are found in [Rav95, HC97]. Here we will restrict us to only defining the DC abbreviations used throughout the paper:

ℓ	$\equiv$	$\int true$	interval length
$\lceil\,\rceil$	$\equiv$	$\ell = 0$	the point interval
$\lceil p \rceil$	$\equiv$	$\int p = \ell \land \ell > 0$	p holds
$\lceil p \rceil^r$	$\equiv$	$\int p = \ell \land \ell = r$	p holds for r time units
$\Diamond D$	$\equiv$	$true\ ;\ D\ ;\ true$	D holds somewhere
$\Box D$	$\equiv$	$\neg \Diamond \neg D$	D holds allways
$D \longrightarrow \lceil p \rceil$	$\equiv$	$\Box \neg (D\ ;\ \lceil \neg p \rceil)$	D is followed by p
$D \xrightarrow{r} \lceil p \rceil$	$\equiv$	$(D \land \ell = r) \longrightarrow \lceil p \rceil$	D leads to p within r time units

Consider a system *SystemD* specified by the DC fomula D. The Z-schema specification of *SystemD* is:

SystemD
UniverseD
D

Where *UniverseD* denotes the universe comprised of a finite set of state variables, rigid variables and operators in which D is interpreted. Composition of schemas is defined iff the basic symbols occurring in both schemas agree on type and arity [Rav95]. The combined system is constrained by the conjunction of the duration formulas.

Further, the Z-schema notation allows multiple instances of a schema specification. A system *SystemND* specifying n instances of *SystemD* is specified by:

SystemND
$s_1, s_2, ..., s_n$: *SystemD*

3 System model

The task described in [SF97] consist of grasping moving objects with a planar two-fingered hand using visual information about the object (see Figure 1).

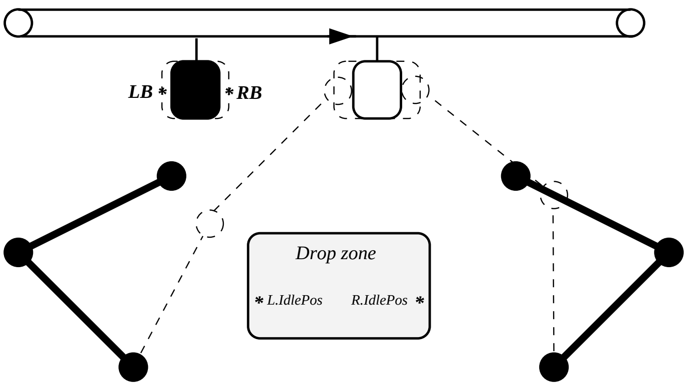

Figure 1: *The grasping robot system*

During the task the fingertips moves from an idle position to the left and right boundary of the object. When both arms have contact with the object it is lifted and moved to a drop zone, where it is released.

The computer system controlling the grasping task is assumed to consist of a finite state machine connected to two robot arms and a task monitor unit. The task monitor is connected to an intelligent subsystem analyzing the visual information from a camera device situated above the system. When an object is present, the subsystem continuously calculates the position of the left and right side of a region in which the object is located and monitors if an undesirable situation occurs . The exact location of the object is not known because the object speed fluctuates as specified below:

Object

$$\begin{array}{lcl} V_{low}, V_{high} & : & \mathbf{R}^{\geq 0} \\ V_{obj} & : & Time \rightarrow \mathbf{R}^{\geq 0} \\ V_{con} & : & \mathbf{R}^{\geq 0} \end{array}$$

$$\square (\lceil \rceil \vee \lceil V_{low} \leq V_{obj} \leq V_{high} \rceil)$$

V_{con} denotes the maximal speed difference between the object and a fingertip when the fingertip makes contact with the object.

3.1 Arm

Because the left and right arm are symmetrical we define them by making two instances of a general arm specification.

Arm

$$\begin{array}{lll} \mathbf{S}, \dot{\mathbf{S}} & : & \mathit{Time} \rightarrow \mathbf{R}^2 \\ \mathbf{B} & : & \mathit{Time} \rightarrow \mathbf{R}^2 \\ \mathbf{IdlePos} & : & \mathbf{R}^2 \\ F & : & \mathit{Time} \rightarrow \mathbf{R}^{\geq 0} \\ \mathbf{V_{reg}} & : & \mathbf{R}^2 \end{array}$$

$$\Box\,(\mathbf{e.S} = \mathbf{b.S} + \textstyle\int \dot{\mathbf{S}})$$

$\mathbf{S}$ denotes the position of the fingertip. $\dot{\mathbf{S}}$ is the speed of the fingertip, which gives the dynamic relation $\Box\,(\mathbf{e.S} = \mathbf{b.S} + \int \dot{\mathbf{S}})$ between $\mathbf{S}$ and $\dot{\mathbf{S}}$. $\mathbf{B}$ denotes the position on the boundary of the object region calculated by the intelligent subsystem. $\mathbf{V_{reg}}$ is an absolute speed of the fingertip corresponding to a relative speed between the fingertip and the object below V_{con}.

F denotes the pressure on the fingertip from the object or an obstacle. F is measured by a force sensor with the following specification:

ForceSen

$$\begin{array}{lll} \mathit{Arm} & & \\ \nu & : & \mathbf{R}^{\geq 0} \\ \gamma & : & \mathbf{R}^{\geq 0} \\ \mathit{contact} & : & \mathit{Time} \rightarrow \mathbf{Bool} \end{array}$$

$$\begin{array}{l} \lceil F > \gamma \rceil \xrightarrow{\nu} \lceil \mathit{contact} \rceil \\ \lceil F < \gamma \rceil \xrightarrow{\nu} \lceil \neg \mathit{contact} \rceil \end{array}$$

The force sensor signals *contact* when the pressure is above a threshold γ. Similar to all other sensors a reaction time of the sensor is assumed. The reaction time of the force sensor is ν. The position of the fingertip is measured by a position sensor:

PosSen

$$\begin{array}{lll} \mathit{Arm} & & \\ \rho & : & \mathbf{R}^{\geq 0} \\ \mathit{atidlepos}, \mathit{atboundary} & : & \mathit{Time} \rightarrow \mathbf{Bool} \end{array}$$

$$\begin{array}{l} \lceil \mathbf{S} = \mathbf{B} \rceil \xrightarrow{\rho} \lceil \mathit{atboundary} \rceil \\ \lceil \mathbf{S} \neq \mathbf{B} \rceil \xrightarrow{\rho} \lceil \neg \mathit{atboundary} \rceil \\ \lceil \mathbf{S} = \mathbf{IdlePos} \rceil \xrightarrow{\rho} \lceil \mathit{atidlepos} \rceil \\ \lceil \mathbf{S} \neq \mathbf{IdlePos} \rceil \xrightarrow{\rho} \lceil \neg \mathit{atidlepos} \rceil \end{array}$$

The arm is moved by altering the angle of the two arm links. When modelling the arm actuator though, we ignore the mechanics and define the actuator as acting directly on the fingertip position:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ArmAct}} \\
\hline
\textit{Arm} \\
\textit{Object} \\
\mu_b, \mu_g, \mu_f, \mu_i, \mu_h \quad : \quad \mathbf{R}^{\geq 0} \\
\textit{movecom} \quad : \quad \textit{Time} \rightarrow \{\textit{Gotoboundary}, \textit{Grasp}, \textit{Follow}, \textit{Gotoidlepos}\} \\
\textit{hold} \quad : \quad \textit{Time} \rightarrow \mathbf{Bool} \\
\hline
\lceil \textit{gotoboundary} \rceil \xrightarrow{\mu_b} \lceil \mathbf{S} = \mathbf{B} \wedge \dot{\mathbf{S}} = (V_{obj}, 0) \rceil \\
\lceil \textit{grasp} \rceil \xrightarrow{\mu_g} \lceil \dot{\mathbf{S}} = \mathbf{V_{reg}} \rceil \\
\lceil \textit{follow} \rceil \xrightarrow{\mu_f} \lceil \dot{\mathbf{S}} = (V_{obj}, 0) \rceil \\
\lceil \textit{gotoidlepos} \rceil \xrightarrow{\mu_i} \lceil \mathbf{S} = \mathbf{IdlePos} \wedge \dot{\mathbf{S}} = \mathbf{0} \rceil \\
\lceil \textit{hold} \rceil \xrightarrow{\mu_h} \lceil F = P \rceil \\
\hline
\end{array}$$

The move commands are given by abbreviations $grasp \overset{\text{def}}{=} (movecom = Grasp)$ etc. . *Grasp* makes the arm enter the object region with a speed of $\mathbf{V_{reg}}$ corresponding to a speed between the fingertip and the object equal to V_{con}. The control signal *Follow* is set when the fingertip makes contact with the object. *Hold* is used when both fingers have contact with the object and is used to lift the object from the feed belt. Combined with *gotoidlepos* it is used to move the object to the drop zone.
The specification of the general arm is a composition of the three components above:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ArmCon}} \\
\hline
\textit{FS} \quad : \quad \textit{ForceSen} \\
\textit{PS} \quad : \quad \textit{PosSen} \\
\textit{AA} \quad : \quad \textit{ArmAct} \\
\hline
\end{array}$$

3.2 Arm Assembly

The arms are linked to the controller through control states defined in the schemas *ArmCom*, *LeftArmCom* and *RightArmCom*. The mapping between the control states and the arm states is specified by the instantiation of the two arms given by:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ArmAss}} \\
\hline
Arm \\
ArmCom \\
ArmLeftCom \\
ArmRightCom \\
L, R : ArmCon \\
\hline
L.\mathbf{V_{reg}} = (V_{obj} + V_{con}, 0) \\
atboundary = L.PS.atboundary \wedge R.PS.atboundary \\
L.FS.contact = lcontact \\
L.AA.gotoboundary = R.AA.gotoboundary = gotoboundary \\
\ldots \\
\hline
\end{array}$$

3.3 Task Monitor

The task monitor receives *ObjectAtImage* from the intelligent subsystem indicating the arrival of an object. During the approach phase the subsystem observes the configuration of the arms. If a critical situation arises the task monitor receives the signal *FailAtImage*.

$$\begin{array}{|lcl}
\multicolumn{3}{l}{\textit{TaskMon}} \\
\hline
TaskMonCom & & \\
ObjAtImage, FailAtImage & : & Time \to \mathbf{Bool} \\
\xi & : & \mathbf{R}^{\geq 0} \\
\hline
\lceil ObjAtImage \rceil \xrightarrow{\xi} \lceil oktograsp \rceil & & \\
\lceil \neg ObjAtImage \rceil \xrightarrow{\xi} \lceil \neg oktograsp \rceil & & \\
\lceil FailAtImage \rceil \xrightarrow{\xi} \lceil failure \rceil & & \\
\lceil \neg FailAtImage \rceil \xrightarrow{\xi} \lceil \neg failure \rceil & & \\
\hline
\end{array}$$

TaskMonCom defines the control states *oktograsp* and *failure*, that links the task monitor to the controller.

3.4 Controller

The controller consist of a finite state machine given by the diagram in Figure 2. The behaviour of the controller is specified by the schemas *Seq*, *ConP*, *Prog*, *Stab* and *Sync*. *ConP* defines the controller phases:

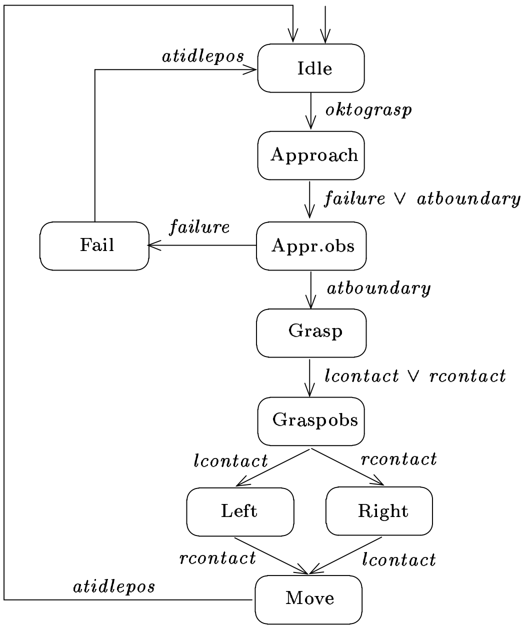

Figure 2: *The controller*

ConP

$$
\begin{array}{lll}
main & : & Time \to \{Idle, Approach, Approachobs, \\
 & & Grasp, Graspobs, Left, Right, Move, Fail\}
\end{array}
$$

The controller phases correspond to the phases shown in Figure 2. The phases are given by abbreviations $idle \overset{\text{def}}{=} (main = Idle)$ etc.

To save space we only specify the *Grasp* and *Graspobs* phases in the remaining schemas. The specification of the rest of the phases can be generalized from this specification. The *sequencing* constraints of *Grasp* and *Graspobs* are:

Seq

ConP

$$
\begin{array}{l}
\lceil grasp \rceil \longrightarrow \lceil grasp \lor graspobs \rceil \\
\lceil graspobs \rceil \longrightarrow \lceil graspobs \lor left \lor right \rceil
\end{array}
$$

When the progression conditions are satisfied progression of the phases happens within δ:

Prog

$ConP$
$ArmCom$
$LeftArmCom$
$RightArmCom$
$TaskMonCom$
$\delta \quad : \quad \mathbf{R}^{\geq 0}$

$$\lceil grasp \wedge (lcontact \vee rcontact) \rceil \xrightarrow{\delta} \lceil \neg grasp \rceil$$
$$\lceil graspobs \rceil \xrightarrow{\delta} \lceil \neg graspobs \rceil$$

Otherwise the phases are stable:

Stab

$ConP$
$ArmCom$
$LeftArmCom$
$RightArmCom$
$TaskMonCom$

$$(\lceil \neg grasp \rceil \;;\; \lceil grasp \wedge \neg(lcontact \vee rcontact) \rceil) \longrightarrow \lceil grasp \rceil$$
$$(\lceil \neg graspobs \rceil \;;\; \lceil graspobs \wedge lcontact \rceil) \longrightarrow \lceil graspobs \vee left \rceil$$
$$(\lceil \neg graspobs \rceil \;;\; \lceil graspobs \wedge rcontact \rceil) \longrightarrow \lceil graspobs \vee right \rceil$$

Finally, the synchronization with actuators happens within η:

Sync

$ConP$
$ArmCom$
$LeftArmCom$
$RightArmCom$
$TaskMonCom$
$\eta \quad : \quad \mathbf{R}^{\geq 0}$

$$\lceil grasp \vee graspobs \vee right \rceil \xrightarrow{\eta} \lceil lgrasp \rceil$$
$$\lceil \neg(grasp \vee graspobs \vee right) \rceil \xrightarrow{\eta} \lceil \neg lgrasp \rceil$$
$$\lceil grasp \vee graspobs \vee left \rceil \xrightarrow{\eta} \lceil rgrasp \rceil$$
$$\lceil \neg(grasp \vee graspobs \vee left) \rceil \xrightarrow{\eta} \lceil \neg rgrasp \rceil$$

3.5 Grasping Robot

Having specified all the parts of the grasping robot, the entire system is specified by their composition:

GRob
TaskMon
ArmAss
Seq
Prog
Stab
Sync

4 Requirements

The requirements consist of assumption and commitment pairs. The assumptions are preconditions to the commitments [HHF$^+$94]. A commitment C with assumption A thus gives the requirement

$$A \Rightarrow C$$

Assumptions are essentially properties which during the design are assumed to be satisfied by the system. Given assumptions A, commitment C and a design D, the verification of the design demonstrates

$$D \Rightarrow (A \Rightarrow C)$$

The system commitments stated in [SF97] are:

- When a critical event occurs the fingers need to be returned to their initial positions:

$$\begin{array}{lll} \mathit{Returns} & \equiv & \lceil \mathit{FailAtImage} \rceil \xrightarrow{T_{ret}} \lceil \mathit{Idlepos} \rceil, \text{ where} \\ \mathit{IdlePos} & \equiv & L.\mathbf{S} = L.\mathbf{IdlePos} \wedge R.\mathbf{S} = R.\mathbf{IdlePos} \end{array}$$

- To guarantee safety of force sensors and object impedance we require that the contact force stays below a certain upper bound:

$$\mathit{ForceCons} \;\equiv\; \Box\,(\lceil\,\rceil \vee \lceil L.FS.F < F_{max} \wedge R.FS.F < F_{max} \rceil)$$

- Total time of the task execution should not exceed a certain time T_{max}:

$$TimeCons \;\equiv\; \lceil \neg atidlepos \rceil \xrightarrow{T_{max}} \lceil atidlepos \rceil$$

The system requirements are thus:

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;Req\;}} \\
GRob \\
\hline
Assumpt_1 \Rightarrow Returns \\
Assumpt_2 \Rightarrow ForceCons \\
Assumpt_3 \Rightarrow TimeCons \\
\hline
\end{array}$$

5 Verification

The design is verified by proving that it satisfies the requirements. In this section we sketch a prove of the first requirement *Returns* by means of the DC proof system described in [Rav95].

It is assumed that *FailAtImage* only can become true in the phase *approach* and stays true until the arms have returned to their idle positions. Further, it is assumed, that *failure* is false when *FailAtImage* becomes true and that *atboundary* is false when *FailAtImage* is true:

$$\begin{array}{rcl}
Assumpt_1 & \equiv & \lceil FailAtImage \rceil \longrightarrow \lceil FailAtImage \vee approach \rceil \\
& & \lceil FailAtImage \wedge \neg IdlePos \rceil \longrightarrow \lceil FailAtImage \rceil \wedge \\
& & \Box\, (\lceil \neg FailAtImage \rceil \;;\; \lceil FailAtImage \rceil \Rightarrow \\
& & \lceil \neg FailAtImage \rceil \;;\; \mathbf{b.}\neg failure) \wedge \\
& & \lceil approach \wedge FailAtImage \rceil \longrightarrow \lceil \neg atboundary \rceil
\end{array}$$

Let $T_{ret} = \xi + 2\delta + \eta + \mu_i$. Using proof by contradiction gives:

$$
\begin{array}{ll}
 & \mathit{true} \ ; \ \lceil \mathit{FailAtImage} \rceil^{T_{ret}} \ ; \ \lceil \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\mathit{Init}\} \\
 & \mathit{true} \ ; \ \lceil \neg \mathit{FailAtImage} \rceil \ ; \ (\lceil \mathit{FailAtImage} \rceil \wedge \ell \geq T_{ret}) \ ; \\
 & \lceil \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\mathit{Assumpt}_1, \ \mathit{Stab}, \ \mathit{Propagation}\} \\
 & \mathit{true} \ ; \ \lceil \neg \mathit{FailAtImage} \rceil \ ; \ (\lceil \mathit{FailAtImage} \wedge \neg \mathit{atboundary} \wedge \\
 & \neg \mathit{failure} \wedge \mathit{approach} \rceil \wedge \ell \geq T_{ret}) \ ; \ \lceil \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\mathit{TaskMon}, \ \mathit{Assumpt}_1, \ \mathit{Stab}, \ \mathit{Propagation}\} \\
 & \mathit{true} \ ; \ \lceil \neg \mathit{FailAtImage} \rceil \ ; \ \ell \leq \xi \ ; \ (\lceil \mathit{failure} \wedge \neg \mathit{atboundary} \wedge \\
 & \mathit{approach} \rceil \wedge \ell \geq 2\delta + \eta + \mu_i) \ ; \ \lceil \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\mathit{Prog}, \ \mathit{Seq}, \ \mathit{Propagation}\} \\
 & \mathit{true} \ ; \ \lceil \neg \mathit{FailAtImage} \rceil \ ; \ \ell \leq \xi + \delta \ ; \ (\lceil \mathit{failure} \wedge \neg \mathit{atboundary} \wedge \\
 & \mathit{approachobs} \rceil \wedge \ell \geq \delta + \eta + \mu_i) \ ; \ \lceil \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\mathit{Prog}, \ \mathit{Stab}, \ \mathit{Propagation}\} \\
 & \mathit{true} \ ; \ \lceil \neg \mathit{FailAtImage} \rceil \ ; \ \ell \leq \xi + 2\delta \ ; \ (\lceil \mathit{fail} \rceil \wedge \ell \geq \eta + \mu_i) \ ; \\
 & \lceil \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\mathit{Sync}, \ \mathit{Stab}, \ \mathit{Propagation}\} \\
 & \mathit{true} \ ; \ \lceil \neg \mathit{FailAtImage} \rceil \ ; \ \ell \leq \xi + 2\delta + \eta \ ; \ (\lceil \mathit{fail} \wedge \mathit{gotoidlepos} \rceil \wedge \\
 & \ell \geq \mu_i) \ ; \ \lceil \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\mathit{ArmAct}, \ \mathit{Propagation}\} \\
 & \mathit{true} \ ; \ \lceil \neg \mathit{FailAtImage} \rceil \ ; \ \ell \leq \xi + 2\delta + \eta + \mu_i \ ; \\
 & \lceil \mathit{IdlePos} \wedge \neg \mathit{IdlePos} \rceil \\
\Rightarrow & \{\text{logic}\} \\
 & \mathit{false}
\end{array}
$$

6 Conclusion

In this paper we have shown that a specification and verification of a complex robotics task based on RL has several advantages: First, DC's capability of specifying hybrid systems integrates the specification of controller program and physical devices. Second, the modularity of Z-schemas makes the system model intuitive reflecting the physical and logical composition of the system, which third, makes the specification easy to alter and divide between several developers.

Further, The generic *ArmCom* Z-schema demonstrates, that RL is very applicative supporting easy specification of component classes.

Acknowledgement

I would like to thank Anders P. Ravn for proposing this work and for invaluable support during the writing process.

References

[HC97] M. R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9(3):283–33, 1997.

[HHF^{+}94] Jifeng He, C. A. R. Hoare, M. Fränzle, M. Müller-Olm, E.-R. Olderog, M. Schenke, M. R. Hansen, A. P. Ravn, and H. Rischel. Provably correct systems. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *LNCS*, pages 288–335. Springer-Verlag, 1994.

[Mos85] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[ORS96] E. Olderog, A.P. Ravn, and J. U. Skakkebæk. Refining system requirements to program specifications. In Constance Heitmeyer and Dino Mandrioli, editors, *Formal methods for Real-Time computing*. John Wiley & Sons, 1996.

[Rav95] A. P. Ravn. Design of embedded real-time computing systems. Technical Report ID-TR: 1995-170, IT/DTU, Lyngby, Denmark, 1995.

[RRH^{+}95] A. P. Ravn, H. Rischel, Holdgaard, Eriksen, and C. Andersen. Hybrid control of a robot - a case study. In *Hybrid Systems*, volume 999 of *Lecture Notes in Computer Science*, pages 391–404. Springer-Verlag, 1995.

[SF97] E. Shkel and N. J. Ferrier. Specifying and verifying visual grasping tasks. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 688–694, April 1997.

[Spi87] J. M. Spivey. *The Z notation.* International Series in computer science. Prentice-Hall, November 1987.

[ZHR91] Zhou, Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Proc. Letters*, 40(5), December 1991.