CRL Specifications
Version 1.2.1

Summary

1. Introduction

The purpose of CRL (common robotic language) is to describe a single or multiple behaviors/actions of a single or of a group of robots. CRL language describes the flow of the control commands and the state of the robot by a structured approach allowing to represent the robot as a set of agents organized in a hierarchical tree. Thus at any time CRL gives precise information about the robot external activity and internal state. Each agent in this tree is able to process a set of commands that is by default the same for each agent. CRL is designed to be used on non-mobile humanoid robots and allows to precisely describe behavior of such a robot (or of a group of robots), and thus is a framework for robot-robot or human-robot interaction.

There already exists a lot of languages for hardware, computer and other robotics related fields. The reason for creating CRL is mainly because we are focusing on features not described by most of the existing languages; features we are interested in are behavioral, psychological and communication features for robots and devices having such possibilities of expression. CRL is powerful enough to describe any type of actions of the behavioral robot and this is mainly because:

- It is designed to be arbitrary modifiable: this means no prescribed set of key words is defined beside the default package controlling the head. This allows to design human-readable behavioral scripts such as theater plays, discussion or just simple non-verbal expression.
- Its grammar is specified by a file (that can define as many rules as the complexity of the robot requires) or the default (built-in) set of commands can be used

- Is multileveled; the language is suitable to describe both a single robot and all its behavior so as multiple robots and their timing, cooperation and coordination of individual activities.
- It is hierarchic, thus allows embedded commands within other commands. As shown later CRL, uses

CRL is 100% compliant with XML thus the parser is either a DOM or a SAX parser as described by the Sun Corporation. The description can be found either by visiting the http://java.sun.com and do a appropriate search or directly on the web site http://www.saxproject.org/ for the SAX parser or http://www.mozilla.org/projects/blackwood/dom/ for the DOM parser.

2. Structure and Behavior

CRL is very simple as in syntax and structure. In general the command has the following form:

<Destination><Command>Parameters

or

<Destination><Command>

but because the language is XML compliant the complete syntax is represented as:

<Destination><Command>Parameters</Command></Destination>.

The 'Destination' tag represents the robot or one of the robot's subpart that is to execute the given command. It can be the key word 'robot' or any other existing (previously created) robot part such as head. The 'Command' tag is the name of the command it self (described later). Finally the 'Parameters' tag represents the possible (if any) parameters of the given command.

The basic CRL is defined for every robot and each robot part so as each element of the robot can be controlled by a set of minimal commands. CRL can be modified by simply inserting tags recursively defined on predefined commands and robot elements (All yet specified tags are at the end of this file). The idea of the CRL is as already mentioned to have a common description of robotics (smart robotics) devices designed for communication and interaction either with humans or with other robots via human like interaction. The CRL is suitable for human reading and thus all tags are expressed in English. This allows in general creating files describing groups or single robots and their activity in time and space so as activity vice versa other robots.

As already mentioned in the introduction, CRL breaks the robot into agents. For example a simple robotic head can be designed with four motor degrees of freedom plus the speech synthesizer. Each device belongs to a higher level hierarchy designed by the user. This higher level is a bio-mimetic representation of the robot. This means that the robot is split into a set of robot elements (RobotParts) that each can contain a set of devices. The robot above will have a *Mouth* that is using one servo to move the jaw and the speech synthesizer. It will also have *Eyes* that will be using two servos (vertical and horizontal movement). Thus the robot is now represented by two higher level agents *Mouth* and *Eyes*, and from five low level device agents.

The general behavior of CRL allows to generate N simultaneous commands, where N is the number of higher level agents. The example form above will be then able to run two simultaneous commands at a time.

CRL is fed to the robot through a CRL parser, that parses all the commands form the script into the memory, and then the robot executes the script in received order.

3. CRL Command Description

The following commands are directly implemented in the CRL parser in order to provide a already functioning robotic platform. There are two global classes of commands (that are explained in the Section ***Single robot scripting***): the direct commands affect the whole robot and the indirect commands are used to address single robotic parts. Among these two classes of commands there are two subgroups: simple commands and complex commands.

a) Simple Commands

By default all given commands are simple. The name comes from the fact that such a command is a single instruction to a single robot part. For example command *<hand><move>12 46</move></hand>* places two servos of the *hand* on the position 12 and 46. Simple commands are the building blocks of CRL.

- The Direct Commands are:
  - *<sync>* - synchronize the actions and object if they are children of this node
  - *<wait> #* - pause the robot for a given amount of time (in seconds). Note, this command does not stop the robot instantaneously but rather waits until all action in progress are finished.
  - *<flush>* - clear all commands in the memory and wait until all parts of the robot terminate their actions.

- The Indirect Commands are
  - *<move> #* - move a given part to position defined in the range 0 – 100. Note this command can be multi-variable for robot parts using more than one servomotor.
  - *<close>* - close the given robot part; put the servo to position MIN
  - *<open>* - open the given robot part; move the servo to position MAX

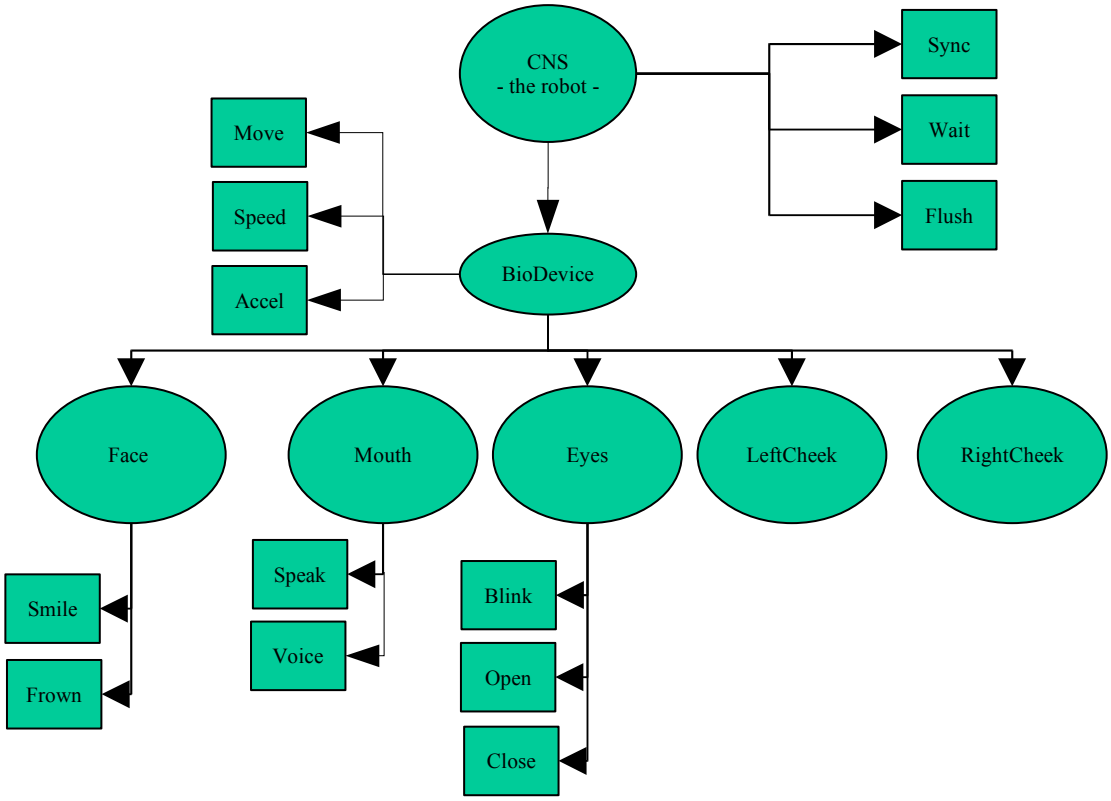
- The Control Commands are
  - *<min>* - the min of the servo range
  - *<max>* - the max of the servo range
  - *<speed>* - the speed of servo
  - *<accel>* - the acceleration
  - *<rate>* - the speed for the speech generator
  - *<react>* - the reactivity of this robot part
  - *<volum>* - the volume for speech and recognition
  - *<voice>* - the voice for the synthesizer
  - *<pitch>* - the pitch of the voice
  - *<waittime>* - the step time for this robot part
  - *<waitstep>* - the size of the step to wait

    - This list is to be extensively extended

- Example of objects (are arbitrary and depends on the user definition) that can be addressed:
    - <*cheek*> - each cheek can be accessed individually
    - <*eyes*> - both eyes move only together
    - <*mouth*>
    - <*eyebrow*> - individual access
    - <*eyelid*> - individual access

- Finally some custom commands are available for specific robot parts and are shown as example:
    - <*smile*># - parameterized smile 0 – 100 (refer to the detail section to check the size and number of parameters).
    - <*frown*># - same as above
    - <*speak*># - speak a sentence or a text with synchronized mouth movements; general tag indicating the sequence is a speech command. (Refer to the JSML web page: http://java.sun.com/products/javamedia/speech/forDevelopers/JSML/ for the syntax defined for the Speech control).
    - <*blink*> - blinks the eyes; note a random blinking is already defined as a automatic feature of the robot

The understanding of how this is working is pretty easy. Each # indicates the parameter in a scale between 0 and 100 where 0 is defined as MIN and 100 is the MAX. The polarity of MIN and MAX can be inversed by the user. Thus the command <*mouth*><*move*>*100*</*move*></*mouth*> results in the biggest possible opening of the mouth, also obtainable using <*mouth*><*open*></*open*></*mouth*>. The order in which the CRL parser is working represents the hierarchy of the robot. The Figure 1 shows the robot structure.



Drawing 1The Command Hierarchy of the CRL robot construct

b) Complex Commands

The second class of commands allows to create macros that can execute a sequence of commands. For example command *smile* can be defined as a sequence of movement, that would generate a happy behavior. Details are described later in this document.

c) Command Rewriting

Another feature of CRL is the command rewriting. For this assume the following example. A command smile was defined in the face object and was parameterized according to some specifications (number of degree of freedom, expression, etc.). Now if someone wants to have different smiles but does not want to create a new command for the simplicity of the script it is possible to parameterize t the smile command by writing is as a normal movement command so as <smile></smile> becomes <smile>#</smile>. In the case the number of parameters is correct the smile command will be executed with the given parameters. In the other case where the parameters do not satisfy the definition of command smile, only parameters fitting the definition will be used and the invalid ones will be discarded.

The commands that CRL accept are mainly for the motor and and control purposes. However CRL also allow to interface to the higher level of the robot. This particular shortcut is is due to the fact that the motor control describes all motor control while the higher level functions are more sensor oriented (e.g. Dance generator, Motion Detector, etc). The syntax for commands to the higher level depends on what module is being used. As in general the robot must be in particular preset on the motor side, the

higher level functions do not require complex command syntax. For example the command dance is defined as:

```
<command>dance
   <parameters>null null</parameters> dance <command><file>
</command>
```

other robot commands are:

```
<command>follow //starts the motion following – requires camera
   <parameters>null null</parameters> //follow <command>
</command>
<command>recog//starts the speech recognition
   <parameters>null null</parameters> //recog <command>
</command>
<command>mood//sets the robot in particular mood
   <parameters>null null</parameters> //mood <command> <params>
</command>
```

4. Robot Constructor

A sub part of CRL is used to describe the current robot and construct a software model of the controller. The constructor file specifies the initial parameters of the robot so as the robot architecture and commands. This file (as any other CRL command script) is defined by an opening tag <*crl*> followed by the <*robot*> defining tag.

```
<crl>
 <robot>robot
```

First direct commands can be declared. Default direct and indirect commands are not used in the constructor as they come by default with the software and are hard encoded. This robot has a single direct command “Dance”

```
<command>dance
     <parameters>null null</parameters>
</command>
```

and with a single robot part “eyes”.

```
<part>eyes
```

Each defining robot part, must include the devices that this object controls. In this case these are Servos 0 and 1 (position as appears on PCB where the Servos are connected). The *device* section may optionally include the range of the Servos such as shown here with the tag *min* and *max*. It can be noticed that the second servo is inverted with respect to the original settings.

```
<devices>0 1
   <min>100 0</min>
   <max>0 70</max>
</devices>
```

Finally the robot comes with a defined acceleration command for the *eyes* robot part. This is a simple command as can be seen by the definition.

```
        <command>accel
          <parameters>null null</parameters>
        </command>
      </part>
    <robot>
  <crl>
```

More complex example is the same robot with complex *init* command. This command allows to set the robot part *eyes* to be set to a particular state: in this case it reset the servo range to 0 to 100 on both servos and set both of them to a speed and acceleration.

```
  <crl>
    <robot>robot
        <command>dance
          <parameters>null null</parameters>
        </command>
      <part>eyes
        <devices>0 1
          <min>100 0</min>
          <max>0 70</max>
        </devices>
        <command>init
          <parameters>speed 50 20 </parameters>
          <parameters>accel 10 10 </parameters>
          <parameters>min 0 0 </parameters>
          <parameters>max 100 100 </parameters>
        </command>
      </part>
    <robot>
  <crl>
```

The command init allows to set the robot part to a desired initial state. It is possible to create more complex commands in order to achieve more complex behavior. An example would be to make the robot part eyes move in a certain way.

```
        <command>coolmove
          <parameters>speed 50 20 </parameters>
          <parameters>accel 10 10 </parameters>
          <parameters>0 0 </parameters>
          <parameters>10 100 </parameters>
          <parameters>accel 10 10 </parameters>
          <parameters>10 10 </parameters>
          <parameters>100 100 </parameters>
        </command>
```

Note that there is one implicit command included in the robot. This command is move. Not only that but the root has a set of implicitly created commands. These

commands are mainly settings but also include the command for move and speech. In general it is good usage to declare move for every robot part in the constructor as follows :

```
<part>neck
.
.
  <command>move
    <parameters>null null null null</parameters>
  </command>
```

Defined commands can be simple such as move above or such as coolmove(above above). The syntax for definition is similar and is explained in table bellow:

Simple Command	***Complex Command***
Single parameter line: Only parameters (e.g. '100 null')	*Multiple Parameter Lines:* Whole command (e.g. 'max 45 50')
Keywords allowed (e.g. Null, ignore)	Value restricted (e.g. '50 23')

The idea behind this definition is to allow to construct scripts for behavior as a single command and also to allow to control it on the fly.

In general all robot parts and commands are equivalent in the way they are build and used but there are some specifications that must be followed. Moreover as all integrated robot parts and devices are designed using the BioDevice/BioRobotPart specifications, they accept but do not understand all the commands. This allows to send wrong commands to the robot and being able to recover.

To use the speech, a robot part containing at least the speech synthesizer and one servo, as they are going to be synchronized during the speech process. (The servo can be moved without the speech). This is also the start of integrating multiple servos in mouth in order to allow tongue, lips, etc movement. The constructor for the robot part mouth is shown below.

```
<crl>
  <part>mouth
    <devices>0 1
      <min>100 0</min>
      <max>0 70</max>
    </devices>
    <command>init
      <parameters>speed 50 20 </parameters>
      <parameters>rate 50 20 </parameters>
      <parameters>accel 10 10 </parameters>
      <parameters>min 0 0 </parameters>
      <parameters>max 100 100 </parameters>
    </command>
    <command>move
      <parameters>null </parameters>
    </command>
```

```
        <command>speak
          <parameters>null </parameters>
        </command>
      </part>
    <crl>
```

The command move takes only one argument as does the command speak. This allow to minimize the size of the CRL scripts .

5. Single robot scripting

The CRL scripting allows to describe behavior for a single robot so as for multiple robots. This section describes how to generate CRL script for a single robot. To control a robot in CRL types of tags are used: direct and indirect.

1. Direct tags describe temporal constraints and commands affecting the whole robot. Direct tags have for consequence time conditions and help to organize on a time scale a complex set of behaviors or actions so as allow to synchronize robot behavior. In general from the robot point of view direct tags are command to the robot as a single entity and thus affect the whole robot.
2. The indirect tags describe directly an action or a behavior of a robot sub-part. Indirect tags can be seen as local commands because they are specifically send to a robotic part such as an arm, mouth or eyes.

The construction of the CRL commands follows the general scheme described in the Introduction, however more complex or simpler commands can be also created. In general, the more complex commands are used as Macros, and allow one level deep recursion in CRL. For example assume, we want to create a command called 'no' for the robot part neck, that represents the robot waving its neck back and for left-to-right and back (representing behavioral negation). One can either write:

```
<neck><move>1023 0</move></neck>
<neck><move>1023 100</move></nedk>
<neck><move>1023 0</move></neck>
<neck><move>1023 100</move></nedk>
```

or a corresponding macro:

```
<neck><no>
        <move>1023 0</move>
        <move>1023 100</move>
        <move>1023 0</move>
        <move>1023 100</move>
<no></neck>
```

that can be invoked by the call *<neck><no><no></neck>* in the CRL script. Also, the robot in general is not equipped with macros that must be designed and defined by the user.

The simple commands (defined by default or non-aggregate commands) allows parameters as shown in the previous example (each line with *move* command specifies where the device should move), while the aggregate commands do not allow parameters.

The CRL specifications are similar such as described in the Introduction, but some restrictions exists on the :

- Tags have no parameters and all variables are between tags (arguments). This means that for example if the robot wants to move left the sequence of commands will be looking something like this: *<head><move>100</move></head>*. In this particular case, the servo position adjustment with respect to the environment is 0 – right, 50 – middle and 100 – left.
- Each tag can represent either an action or an object.

Because we are using the JSAPI (Java Speech API) and JSML (Java Speech Markup Language) and it already has a parser each time the tag <speech> is encountered by the parser the tag is modified into *<jsml>*. This is mainly done in order to increase the understandability of the script that might get pretty long. However, the usage of JSML is restricted so far.

The behavior defined by a CRL script is analyzed in similar top-down hierarchy to the picture. That is when a tag is met, the issued command by the parser is sent directly to the robot, which first checks if the tag is a direct command. If not the tag is analyzed in order to determine if it is a valid robot part. If yes the subsequent command is sent to the robot part which either executes it or not. The following example shows the idea.

- *<crl>* - tag defining CRL language
- *<robot>* - tag defining a robot device. All commands inside of this tag will be sent to one unique robot
- *<sync> </sync>* - end of synchronized block – this tag waits until all activity of the robot is done and only then allows next command to be executed.
    - *<face>* - *tag* defining a motor sequence of commands
        <smile> 50 </smile> - tag defining a complex facial expression
        <normal></normal> - tag defining the resting facial expression
    - *</face>* - end of commands to the face
    - *<eyes>*
        <move>50</moves>
    - *</eyes>* - tag defining the movement of the left eye
    - *<mouth>* - mouth is being addressed
        <speech> - beginning of the speech
        This a short paragraph JSML
        </speech> - end of speech
    - *</mouth>*
    - *<eye>* - commands for eye
        - *<blink> 1</blink>* - here left is defined as the left eye and will blink
    - *</eye>*
    - *<face>*
        - *<smile>10</smile>* - once the blinking is over the robot will smile
    - *</face>*
    - *<mouth>*
        - *<speech>* - then speech is initiated
        - My name is cynthea,
        - *</speech>*
    - *</mouth>*
    - *<wait>5 </wait>*

- *<mouth>*
  - *<speech>* - then speech is initiated
  - and i am kinda happy
  - *</speech>*
- *</mouth>*
- *<wait>5</wait>*
- *<mouth>*
  - *<pitch>* - then speech is initiated
  - 60
  - *</pitch>*
- *</mouth>*
- *<mouth>*
  - *<speech>* The cost of my construction is $50000
  - *</speech>*
- *</mouth>*

- *</robot>* - end of script for this robot
- *</crl>* - end of script file

If you went through t the above example and the JSML documentation you noticed the presence of *<wait>* inside of the JSML script. This means that without really modifying the JSML syntax we inserted time precision in order to give more freedom to the CRL. In this particular case the resulting action will be: "My name is Cynthia", break for 5 seconds and "and i am kinda happy". The wait command in here is context independent so as the wait command will be executed no matter what the other actions of the robot are.

6. Multiple robots scripting

Let's have a look on another example illustrating more in details the commands *<sync>* and *<wait>*.

- *<crl>*
- *<sync>* - indicates the sequence of synchronized commands
  - *<robot>1* – indicating the commands will be issued for the robot #1
    <sync> - indicates the following sequence of commands is synchronized
    <eyes> - eye movement
    - *<move>50</move>*

    </eyes>
    <mouth><open></open></mouth>
    </sync> - end of synchronized block inside for this robot
  - *</robot>*
  - *<robot>2* – the second robot command sequence
  - *<mouth>*
    <speech> - speech init
    <sayas> Hello this is not an exercise</sayas>
    <wait>2</wait>
    </speech>
  - *</mouth>*
  - *<sync>* - begin a new synchronized command sequence inside of the robot 2

<eyelid> - begin the move sequence

- *<both><close></close></both>*

</eyelid>

<mouth>

- *<speech>*where are you going ?
- *</speech>*

</mouth>

- *</sync>*
- *</robot>* - end of second robot command sequence

- *<sync>* - end of the synchronized group action
- *<crl>*

This example illustrates the other properties of CRL. First, from both of examples it is obvious no indirect tags can be mixed so as no incoherent commands can be created. Each time an action or a speech commands sequence is introduced in the script the sequence must be finished before another one is described. To synchronize these actions the keyword <sync> is used. Second, there is a strict order of tags. This means that a CRL script can be a sequence of commands for a Single robots, a synchronized sequence of commands for a single robot or a synchronized script for multiple robots as shown in example 2. Other side features apparent are: the movements are parameterized on a scale between 0 and 100; a eye movement with value of 50 will bring the eye in the middle while 0 and 50 will be extremities of its movement. The wait commands parameters are in seconds; we assume that with the current technology used for this project this is precise enough with regards to the hardware and software speed.

In Development

- Relative parametrization – allow partial coordinates to be parsed correctly
- Sensory extension – CRL implementation of the Sensor command flow