basAR CR

Behavioral Authoring System for Augmented Reality with Cross-Reality

http://www.ckirner.com/basar/
https://sites.google.com/site/christophercerqueira/projetos/ear/basar

Writer	Christopher Shneider Cerqueira
	christophershneider@gmail.com
Supervisor	Prof. Dr. Claudio Kirner
	ckirner@gmail.com
Document Type	manual
Subject	basAR
Date	12/04/2011
Revision	

Summary:

Acronyms:

basAR – “Behavioral Authoring System for Augmented Reality”

VRML - “Virtual Reality Modeling Language”

3D – Three-Dimensional

INTRODUCTION:

basAR is an Augmented Reality Environment to create and use applications where behavior can be programmed to enhance the augmented experience. Some application examples are puzzles, building guides, schooling, games, etc.

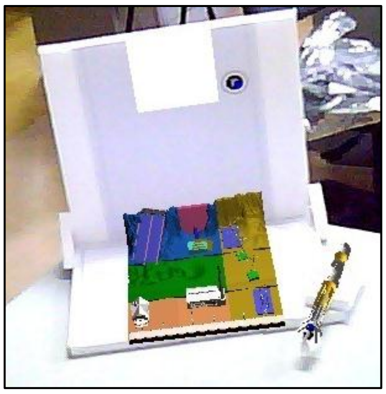

Figure 1 - Puzzle

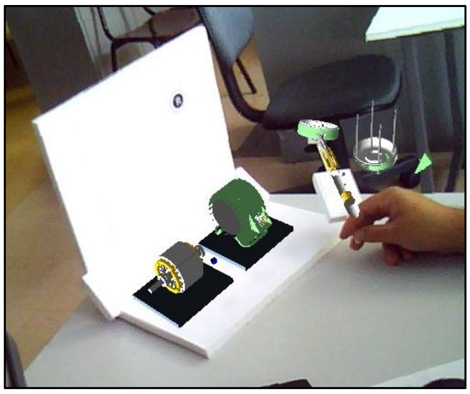

Figure 2 - Building guides

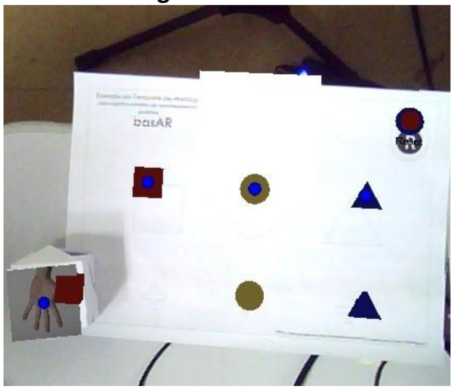

Figure 3 - Children games

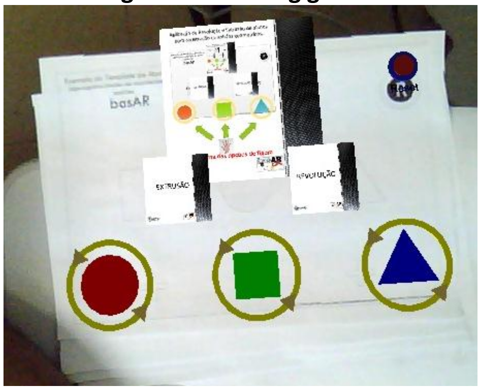

Figure 4 - Schooling applications

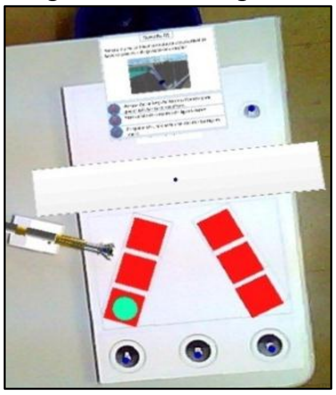

Figure 5 - Games

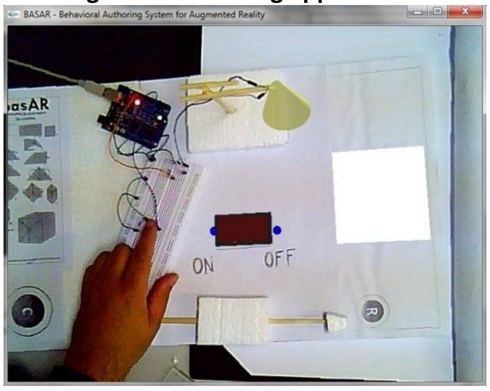

Figure 6 - Cross-Reality

HOW TO USE AN BASAR APPLICATION

basAR applications are based on a concept of action points. Action points are the center of reactive zones from the Structure Layer that are placed relatively of Infrastructure Layer object, in the Figure 7 example, the base Marker. The Actuator Layer object, the Actuator Marker, also has an action point. The virtual collision between a structure action point and an actuator action point, or structure action point and another structure action point carried by the actuator action point provokes a reaction. On basAR the collision reaction or collision feedbacks can be dynamically configured on the authoring mode.

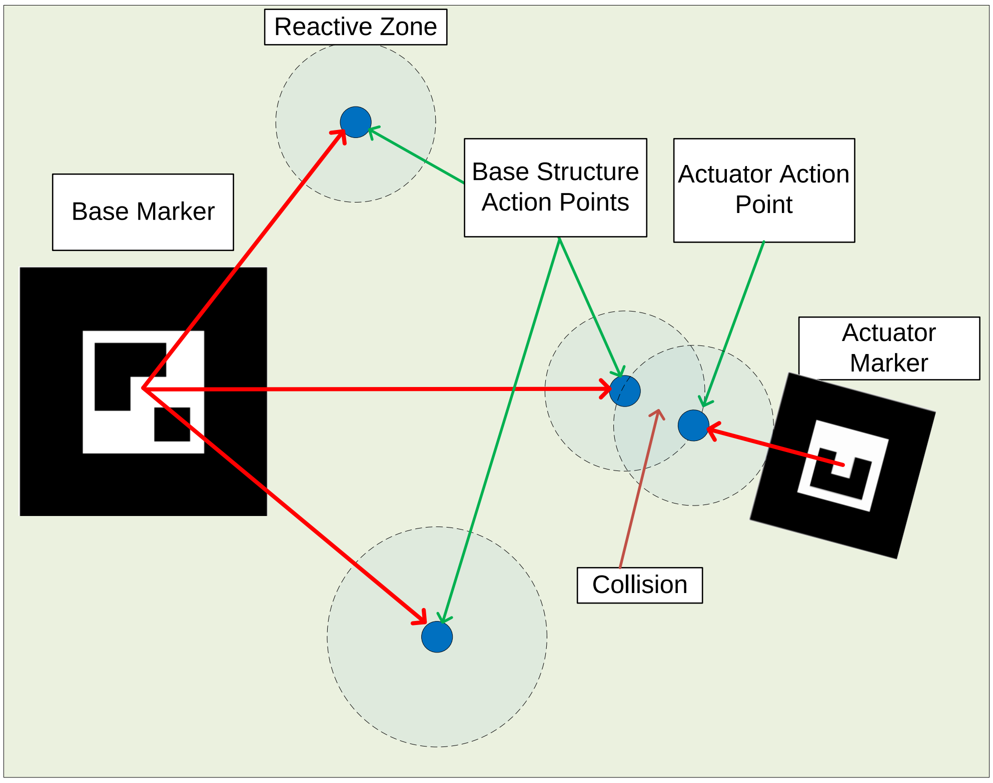

Figure 7 - How basAR works.

The system`s authoring is done by configuration files that are read by the basAR software. Using basAR the augmented reality application doesn`t need to be programmed using a computer language as C/C++, Java, etc., the objects and the behavior are read from configuration files.

The following sections describe how the basAR read those configuration files, how to create them.

ELEMENTS ORGANIZATION:

The system is based on the interaction of action points. Action points are virtual points that are added to the real environment based on infrastructure information, as marker detection. The action points can perform some attitudes or give information depending on how it is set on the environment. To interact with those action points its necessary an actuator device to be recognized to provide information of the user position.

The organization of the elements to achieve this interaction is shown at Figure 8, where the elements are organized into layers. These layers split the development task, in order to organize the work. The Infrastructure layer specifies the workspace, the Structure the reactive zones and augmented scene that is created above the Infrastructure, the context are the scene objects that are used on the application, the Actuator is the method used to interact with the Structure objects and the Behavior is the interaction rules between the Actuator and the Structure and the feedback control of these interactions.

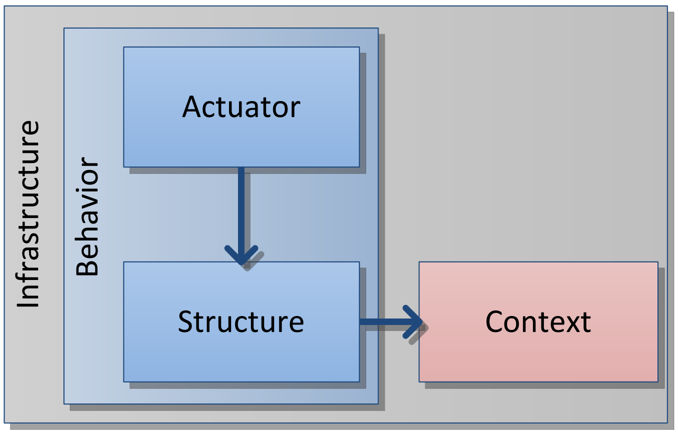

Figure 8 - Layer Organization

The basAR layers organization can be divided into several objects; those objects can be seen at Figure 9. There is a basAR object that connects to the Behavior object; the basAR object calls the basic augmented application objects. The Behavior object controls the action between the Actuator and Base Object, and generates several feedbacks to the user. The Actuator objects defines how the user interact, at this point it's only possible ARToolKit marker interactions. The base Object is responsible to define the Structure layer, several reactive zones, or action points, are created to include objects to the augmented scene or define behavior to a specific reactive zone.

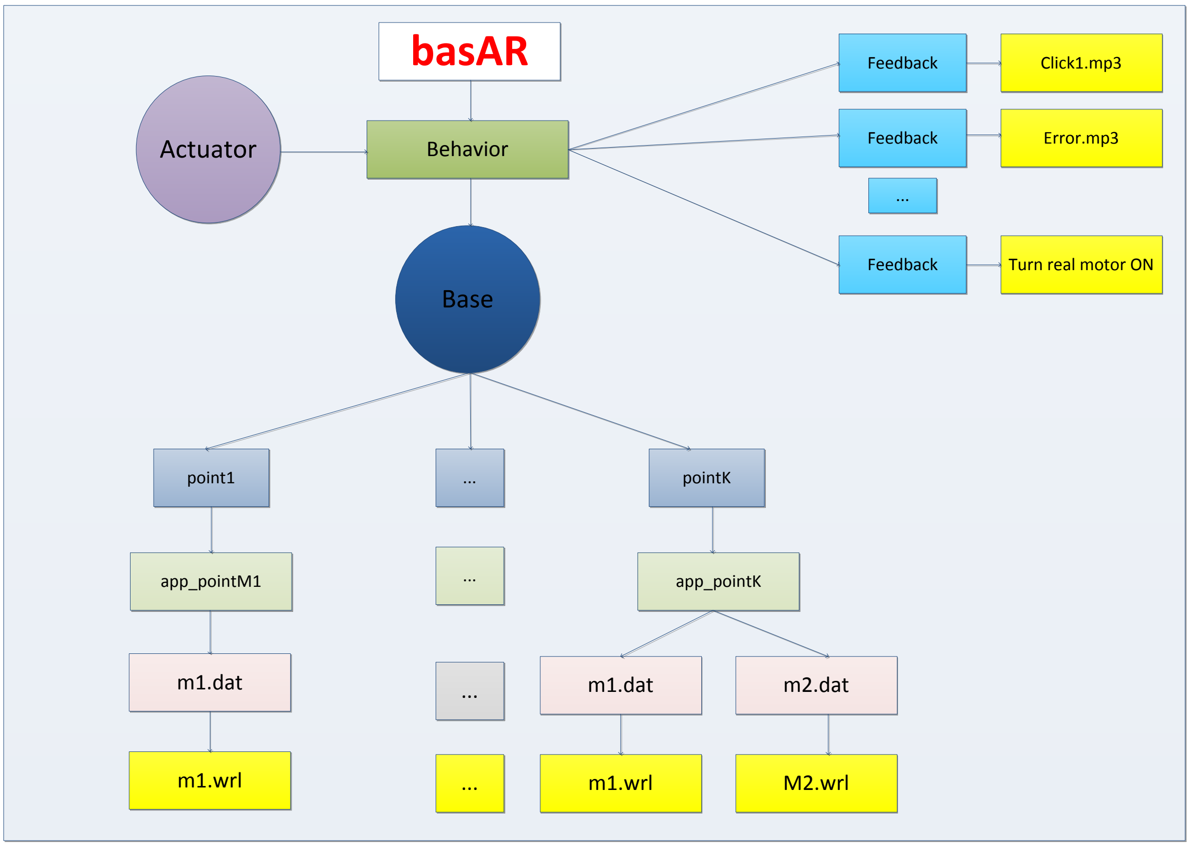

Figure 9 - Objects Organization

The basAR objects and layers are described on several configuration files, which are read on the basAR start-up. Next Section will describe these configuration files, in order to setup the augmented environment.

CONFIGURATION FILES:

When the basAR is started, it read some configuration files to compose the augmented application. These files are organized as show at Figure 10.

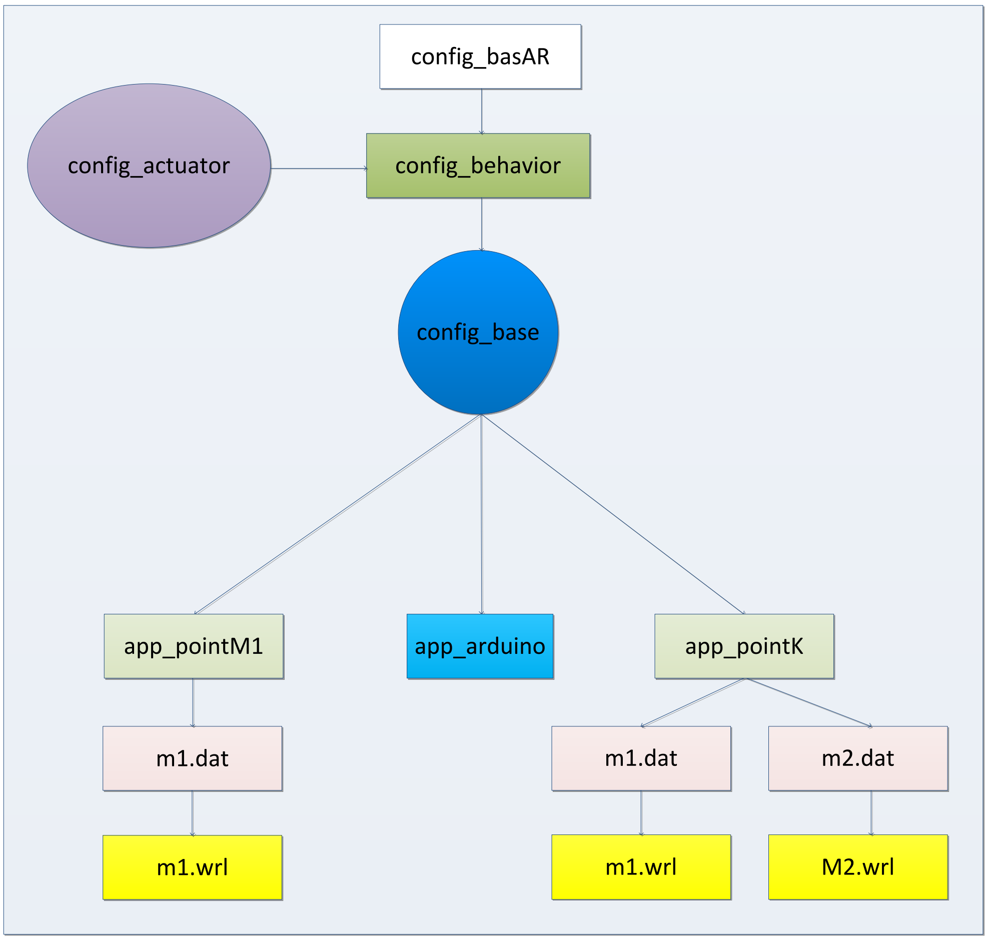

Figure 10 - Files Organization

- **config_basar**: configures the Generic data, and the amount of bases and actuators that can be used.
- **config_actuator**: configures the actuator that the user will make interaction.
- **config_base**: configures the infrastructural source of data to setup a base to virtual data, it is also configured the action points and some of their attributes and configuration files.
- **app_pointX**: configure the objects that will be loaded on the action point.
- **modelX**.dat: configure the object.
- **config_behavior**: configures the interaction between the actuator and action points of the base.
- **config_arduino**: configures an extern connection, designed to connect to ARDUINO.

The descriptions of each configuration file are on the following subsections. The sequence of the configurations files must be respect as the software read it on the order proposed.

CONFIGURE SYSTEM: CONFIG_BASAR

This file configures the standard definitions, background sound and opening sound, the file where is the interaction state machine, the bases and the actuators files.

Field	Entry type	Definition
AppName	String (Uses all line)	Defines what name will appear at application window. The all line (256 characters) will be used as the application name.
ScreenSetup	WINDOWED FULLSCREEN <Width> <Height> <BitDept> <RefreshRate> PROJECTION <Width> <Height> <BitDept> <RefreshRate>	Defines the main screen. • Windowed: Opens the camera view as a window. • Full screen: Opens the camera view on full screen mode. Usually a configuration is FULLSCREEN 600 480 32 0 • Projection: Opens the camera view on full screen mode, however the camera data isn´t shown, only the virtual content. Usually a configuration is PROJECTION 600 480 32 0
StdAwayModel	<TypeModel> <ModelFileAddress>	Standard model to show the action point. When using SENSE_PROX function it represents that the point has no actuator close “requesting” an action. Need to specify the type model: • VRML - to use openVRML library Need to specify the file address to the file that will be loaded to the model library.
StdCorrectModel	<TypeModel> <ModelFileAddress>	Standard model to show on the action point that the action is correct, when using SENSE_PROX function. Need to specify the type model: • VRML - to use openVRML library Need to specify the file address to the file that will be loaded to the model library.
StdWrongModel	<TypeModel> <ModelFileAddress>	Standard model to show on the action point that the action is wrong, when using SENSE_PROX function. Need to specify the type model: • VRML - to use openVRML library Need to specify the file address to the file that will be loaded to the model library.
StdMarkerCover	<TypeModel> <ModelFileAddress>	Standard marker cover. It can be used on a base or actuator that tracks a marker. Need to specify the type model:

		• VRML - to use openVRML library Need to specify the file address to the file that will be loaded to the model library.
StdErrorSound	<AudioFileAddress> <Volume> NO_ERRORSOUND	Standard error sound. If you don't want to read an error sound use NO_ERRORSOUND. Need to specify the source audio file address. Check the chapter about the audio engine used to see the type of files allowed. Need to specify the volume; it goes from 0.1 to 1.0.
BackTrack	<AudioFileAddress> <PlayType> <Volume> NO_BACKTRACK	Backtrack audio. If you don't want to read a backtrack sound use NO_BACKTRACK. Need to specify the source audio file address. Check the chapter about the audio engine used to see the type of files allowed. Need to specify the play type (LOOP, ONCE) Need to specify the volume; it goes from 0.1 to 1.0.
StartSound	<AudioFileAddress> <PlayType> <Volume> NO_STARTSOUND	Audio that will be played at the beginning of the application. If you don't want a start sound use NO_STARTSOUND. Need to specify the source audio file address. Check the chapter about the audio engine used to see the type of files allowed. Need to specify the play type (LOOP, ONCE) Need to specify the volume; it goes from 0.1 to 1.0.
StateMachine	<RulesFileAddress>	Behavior file address to be read to setup the action point's behavior.
NumBases	Number	Number of bases that will be used. Use 0 if no base.
Bases	<BaseFileAddress>	Base file address to be read to specify the base. It need to ready as much files as specified by the NumBases field.
NumActuators	Number	Number of actuators that will be used. Use 0 if no actuator.
Actuator	<AcuatorFileAddress>	Actuator file address to be read to specify the actuator. It needs to read as much files as specified by the NumActuators field.

Example:

```
BASAR – Behavioral Authoring System for Augmented Reality
WINDOWED

VRML wrl/action/ballBlue.dat
VRML Wrl/Action/ballGreen.dat
VRML Wrl/Action/ballRED.dat
VRML Wrl/action/tampa.dat
Audio/explosion.wav 0.5

Audio/backTrack.mp3 LOOP  0.3
Audio/bell.wav      ONCE  0.5

Data/config_behavior

1
Data/config_base

1
ARTKSM Data/config_actuator
```

CONFIGURE ACTUATOR TO INTERACT WITH THE APPLICATION: CONFIG_ACTUATOR

This file configures an actuator source of ARToolKit type from Actuator Layer: the symbolic model that is used to represent the interaction, the action point model, position and action radius.

Field	Entry type	Definition
ActuatorName	String	Actuator name
If using ARToolKit Single Marker Source		
PatternSource	<PatternFileAddress>	ARToolKit pattern file address. Can be created by mk_patt.exe file on BASAR root folder.
PatternWidth	<Width>	Size of the printed pattern, on millimeters
PatternCenter	<X> <Y>	Place where is the 2D center of the pattern.
PatternCover	<TypeModel> <ModelFileAddress>	Actuator pattern cover. Need to specify the type model • VRML - to use openVRML library Need to specify the file address to the file that will be loaded to the model library. Or use USE_DEFAULT to get default cover or NO_COVER to don't use a cover.
	USE_DEFAULT	
	NO_COVER	
SymbolicModel	<TypeModel> <ModelFileAddress>	Model that represent an abstract interpretation of the actuator. Need to specify the type model • VRML - to use openVRML library Need to specify the file address to the file that will be loaded to the model library. Or use NO_SYMBOLIC to don't use.
	NO_SYMBOLIC	
PointModel	<TypeModel> <ModelFileAddress>	Actuator action point model. It is put on the action zone of the actuator. Need to specify the file with the 3 models. Or use DEFAULT_IPOINT to get default value.
	DEFAULT_IPOINT	
PointPosition	<X> <Y> <Z>	3D point coordinates from the center of the pattern where the actuator action point will be. (Millimeters)
PointRadius	<Radius>	Zone around the action point that is reactive. (Millimeters)

Example:

```
ARTKSM1

Data/Markers/pa.patt
37.0
0.0 0.0
NO_COVER

VRML wrl/Action/pa.dat

DEFAULT_IPOINT
20.0 0.0 0.0
400.0
```

CONFIGURE ARTOOLKIT BASE OF VIRTUAL CONTENT: CONFIG_BASE

This file configures the Infrastructure Layer source of the base where the virtual content is loaded, the specific base sounds and the Structure Layer objects: the action points.

Field	Entry type	Definition
BaseName	String	Base name, spaces are not allowed.
If using ARToolKit Single Marker Source		
SourceType	<SourceType>	Tracking source • ARTKSM – ARToolKit Single Marker tracking.
PatternSource	<PatternFileAddress>	ARToolKit pattern file address. Can be created by mk_patt.exe file on basAR root folder.
PatternWidth	<Width>	Size of the printed pattern, on millimeters
PatternCenter	<X> <Y>	Place where is the 2D center of the pattern.
PatternCover	<TypeModel> <ModelFileAddress>	Base pattern cover. Need to specify the type model • VRML - to use openVRML library Need to specify the file address to the file that will be loaded to the model library. Or use USE_DEFAULT to get default cover or NO_COVER to don't use a cover.
	USE_DEFAULT	
	NO_COVER	
VisibleSound	<AudioFileAddress> <Volume>	Patter visible audio. If you don't want use NO_VISIBLESOUND. Need to specify the source audio file address. Check the chapter about the audio engine used to see the type of files allowed. Need to specify the volume; it goes from 0.1 to 1.0.
	NO_VISIBLESOUND	
ErrorSound	<AudioFileAddress> <Volume>	Base error sound. Need to specify the source audio file address. Check the chapter about the audio engine used to see the type of files allowed. Need to specify the volume; it goes from 0.1 to 1.0. If you don't want an error sound use NO_ERRORSOUND, or to use the standard use USE_DEFAULT
	NO_ERRORSOUND	
	USE_DEFAULT	
NumPoints	<number>	Number of action points of the base.
Action Point structure – The following structure repeats to each internal action point		
PointName	String	Point name, spaces are not allowed.
ActionModel	<ModelsFileAddress>	Action point model. It is put on the action zone

	DEFAULT_IPOINT EXTERNAL_IPOINT	of the base. Need to specify the file with the 3 models. Or use DEFAULT_IPOINT to get default value. Use EXTERNAL_IPOINT to setup external hardware connection.
ObjectModels	<ModelsFileAddress> NO_OBJECT <ExternalCommandTable>	Object models associated to the action point. It is put on the action zone of the base. Need to specify the file with the models. Or use NO_OBJECT to any object. If using external point, must allocate file with commands.
StartTranslation	<X> <Y> <Z>	Translation from the center of the base of the action point. (Millimeters)
StartRotation	<X> <Y> <Z>	Rotation of the three axis from the center of the translated action point (Degrees)
StartScale	<X> <Y> <Z>	Scale on the three axis.
StartRadius	<Radius>	Zone around the action point that is reactive. (Millimeters)

Example:

```
BASE1

ARTKSM
Data/Markers/base.patt
74.0
0.0 0.0
USE_DEFAULT
Audio/bell.wav     ONCE  0.5
Audio/explosion.wav 0.5

2

Pen
DEFAULT_IPOINT
Data/config_pointA
-100.0 -100.0 0.0
15.0 0.0 0.0
1 1 1
900.0

attractPen
DEFAULT_IPOINT
NO_OBJECT
-100.0 -100.0 0.0
15.0 0.0 0.0
1 1 1
900.0
```

CONFIGURE LIST OF OBJECTS OF AN ACTION POINT: APP_POINTX

This file contains the list of objects that are associated to the action point. The same structure of file is used to read the action point model of the actuator action point and the base action point.

Field	Entry type	Definition
NumObjects	<number>	Number of objects that on the file
Object structure – The following structure repeats to each object.		
Object	<Type> < TypeModel> <FileAddress>	Define the object that will be loaded to the action point. Need to specify the type of the object • MODEL3D – Model type 3D ○ VRML – Model type 3D, use openVRML object. Need to specify the file that setups the object

Example:

```
3

MODEL3D VRML Wrl/model1.dat

MODEL3D VRML Wrl/model2.dat

MODEL3D VRML Wrl/model3.dat
```

CONFIGURE AN OPENVRML OBJECT FILE: OBJECT.DAT

This file configures the VRML model and it's characteristics to be loaded by the openVRML library.

Field	Entry type	Definition
WRLFileAddress	<FileAddress>	VRML file to the openVRML library, usually a .wrl type of file.
Translation	<X> <Y> <Z>	Translation on the three axis (Millimeters)
Rotation	<X> <Y> <Z>	Rotation on the three axis. (Degrees)
Scale	<X> <Y> <Z>	Scale on the three axis.

Example:

```
deskStuff/pen1.wrl
0.0 0.0 00.0
0.0 0.0 0.0
0.25 0.25 0.25
```

CONFIGURE LIST OF EXTERNAL COMMANDS: APP_SERIAL

This file configures a lookup table to send command to external hardware. Can be used numbers from 10 to 254. The numbers from 00 to 09 and 255 are reserved.

Field	Entry type	Definition
ComNumber	<ComID>	COM Number that is used to send message to the hardware associated to this point.
Object structure – The following structure repeats to each object.		
Command	<RequestNumber> <RequestName> <NextState>	Commands to be sent to the hardware: • RequestNumber: actual value that will be sent to the hardware • RequestName: associated name to be used on behavior configuration • NextState: On reading commands, the thread started to read the hardware, if the waited command is received, it changes to the assigned state

Example:

```
COM4

10 TurnLightON
11 TurnLightOFF

20 isButtonPressed 5
```

CONFIGURE BEHAVIOR: BASAR_BEHAVIOR

The behavior configuration file has a simple structure based on three basic types: BEGIN_STATE, actions, END_STATE. Each type has different possibilities to compose the application behavior. The section X explains how the behavior works and the possible actions.

Field	Entry type	Definition
This structure repeats at each state		
BEGIN_STATE	BEGIN_STATE <StateNumber>	This defines that beginning of the state. It needs to setup the state number. Obs.: It's not necessary to put the states in order, as this state number is given on an action or on the transitory state.
Actions	The actions are explained at Behavior Control Section	
END_STATE	END_STATE	There are two types of states: Working and transitory. The working states waits an action is completed to change to other state, and a transitory state automatically changes to another state after read the actions. The configuration state can be used to configure the action points. To set a working state use only END_STATE. To set a configuration state use END_STATE **GO_TO** <NextState> To delay to the next state can me used the sentence AFTER <Time>, with working and configuration state.
	END_STATE GO_TO <NextState>	
	END_STATE GO_TO <NextState> AFTER <Time>	

Example:

```
BEGIN_STATE 1
            1 STAT ONLY_BALL Audio/explosion.wav
            2 DRGF ONLY_OBJECT
            3 DRGF ONLY_OBJECT
            4 ATTO 2 ONLY_BALL 2 Audio/bell.wav
END_STATE

BEGIN_STATE 2
            2 SCL 2 2 2
            3 SCL 2 2 2
END_STATE  GO_TO 4

BEGIN_STATE 4
            2 STAT ONLY_OBJECT
            3 STAT ONLY_OBJECT
END_STATE
```

BEHAVIOR CONTROL:

The basAR permits to control the behavior between the actuator action points and the bases action points. The behavior's setup can be treated as an interaction state machine, as on Figure 11, where there are input events, an interaction test and the output indicating the next state. The behaviors are configured inside the state machine by commands, as on Figure 12. These commands can be divided in action commands that expect and do a specific behavior as attract or repel points or play audio and configuration commands that configure the action point's attributes.

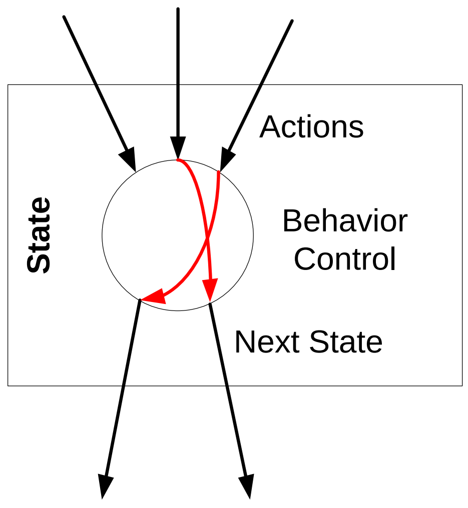

Figure 11 - State

```
BEGIN_STATE 1
        1 STAT ONLY_BALL Audio/explosion.wav
        2 DRGF ONLY_OBJECT
        3 DRGF ONLY_OBJECT
        4 ATTO 2 ONLY_BALL 2 Audio/bell.wav
END_STATE
```

Figure 12 – State described in commands

The association between states configures the application behavior giving more user experience on the augmented reality application.

A) BEHAVIOR BETWEEN ACTION POINTS

The interaction between actions points define the behavior that the application can make. An interaction can be divided in three steps: selection, manipulation and release. So, the behavior consists in selecting an action point, the user mainly action is move the action point or change properties and release it someplace.

The action point selection is the act of the actuator to grab the point. This selection is enabled by the basAR.

The main manipulation is to move the action point, after selecting it. Other manipulation consists in configuration features that can change the action point characteristic.

The action point release can be controlled by the system or by the user. On a reactive area, where action points are set to interact with the moving point can adapt the way that the action point is released. Out of this reactive area, the action point is placed where it's released.

The Figure 13 shows the selection, manipulation and release options on basAR.

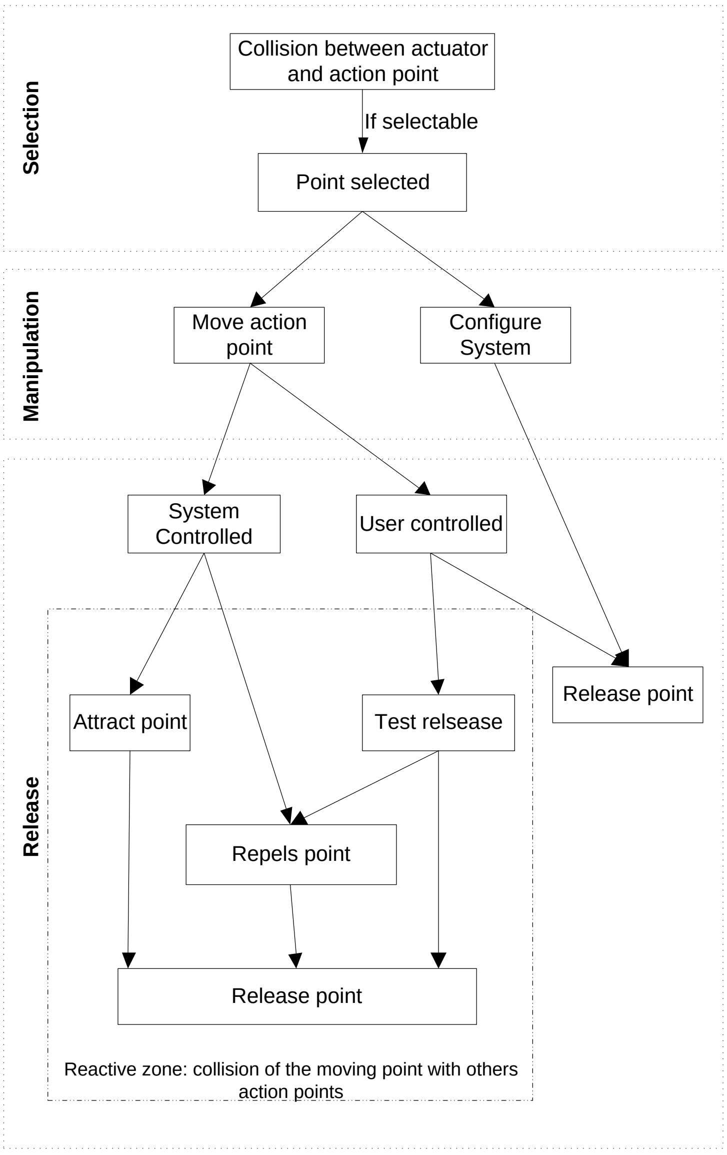

Figure 13 – Action flow.

SHOW MODE

On basAR it's possible to define how the objects from the action point are shown. There are 2 types of objects associated to the action point:

- Action ball: The 3D object model that is placed on the center of the reactive area of the action point.
- Objects: The object that is placed around the action point that move with the action point, and represents an abstraction of the context of the application.

The possible options are:

- HIDE: All objects are hided.
- ONLY_BALL: Only the action ball is shown.
- ONLY_OBJECT: Only the active object is shown.
- BOTH: The action ball and the active object is shown
- FLASH_BALL: Only the action ball is shown and it is flashing.
- SENSE_PROX: Only the action ball is shown and uses the proximity sensor function.
- ALL_OBJECTS: Show all objects.

The HIDE, ONLY_BALL, ONLY_OBJECT, BOTH and FLASH_BALL are quite obvious, but the SENSE_PROX requires extra observations:

1. The SENSE_PROX show mode it's only applied to working behavior.
2. The proximity sensor starts to indicate, when the distance is twice the distance of the point radius.
3. Each behavior treats the SENSE_PROX differently.
4. Some behaviors require 3 or 2 models to indicate proximity. The point can use the default points or indicate a file with 3 or 2 models. If you try to use a behavior that requires 3 models and you only have assigned 2, it will automatically use the default point.
5. The file that indicates the models must have the following sequence
    a. To 3 models: away model, correct behavior model, incorrect behavior model.
    b. To 2 models: away model, correct behavior model.

TRACKING THE ACTION POINT POSITION

Each action point has three positions to be used by the software: the start position, the last position and the actual position.

- The start position is set on the start of the software and is read on the Base configuration file.
- The last position is set when the actuator grabs the action point and starts to move it.

- The actual position is the actual place where the action point is.

Some behaviors use the start position to repel an action point. To change it to use the last position, or a new position you have to use configuration states and configuration commands.

AUDIO ON BEHAVIOR

On some behaviors are possible to set confirmation audio, so, when the behavior is completed correctly an audio is player by the audio library. On the behavior command it's possible to set if the audio will be played over an audio from other state or will stop other audios and play the actual command audio.

Each behavior treat the audio differently, so on each command will be explained the reason that makes the audio to be played.

- STAT: When the actuator pass by an action point with this behavior plays an audio.
- DRGF: When an actuator grabs an action point it plays an audio
- DRGRP: When an actuator grabs an action point with this behavior it plays an audio. If occurs a collision it plays the base error sound.
- ATTO: When an action point moved by an actuator is attracted by an action point with this behavior it plays an audio.
- ATTRP: When an action point moved by an actuator is attracted by an action point with this behavior it plays an audio, otherwise if the action point isn't the expected plays an error sound.
- ATTA: When any action point is attracted by the point with this behavior plays an audio.
- DRPO: When a point moved by the actuator is correctly deposited on the action point with this behavior it plays an audio, if the action point isn't the expected it plays the base error sound
- DRPA: When any action point is deposited on the action point with this behavior it plays an audio.
- RPLO: When an action point moved by an actuator is repelled by an action point with this behavior it plays an audio.
- RPLA: When any action point is repelled by an action point with this behavior it plays an audio.
- CHGST: When an actuator grabs a point with this behavior it plays an audio.

BEHAVIOR IN COMMANDS

In order to setup the system behaviors the basAR provide a command structure to inform the attitude to be taken and the configuration to make. Theses commands are divided in:

- Moving commands

- Attract commands
- Repel commands
- Drop commands
- Configuration commands
- Static command

All the commands are preceded by the base and action point ID, to identify in which action point the attitude or configuration will take place.

Some commands have a NextState field. If the function is correctly completed it changes to the state specified by the command.

Obs.: Command fields on red box are obligatory and on blue are optional.

MOVING COMMANDS

The moving commands enable the action point to be grabbed by the actuator.

DRGF (ENABLES TO DRAG FREELY)

This command enables the base action point to be moved freely by an actuator, if the actuator is already moving other action point and they pass near this point doesn't occur collision.

Point ID	DRGF	ShowMode	Audio	Overplay

Example: 1 DRGF ONLY_OBJECT Audio/walking.mp3 OVER

Proximity sensor behavior: Shows that can be touched. Uses CANWORK model.

Audio reason: Plays while is moving.

DRGRP (ENABLES TO DRAG FREELY)

This command enables the base action point to be moved freely by an actuator, but if the actuator that is already moving another action point pass near this point it provokes a collision, repelling the moving action point to the start position and the state can be changed.

Point ID	DRGRP	ShowMode	NextState	Audio	Overplay

Example: 3 DRGRP ONLY_BALL 15

Proximity sensor behavior: If actuator it´s not moving a point the sensor shows the CANWORK model. If the actuator is moving a point and collides with this point, it shows the CANNOTWORK model.

Audio reason: Plays while is moving. If the actuator is moving a point and collides with this point plays the base error sound.

ATTRACT COMMANDS

Attraction is an away to release the point of its moving. When an action point is moved by the actuator and it collides to an action point that is set to attract, it releases the point and it copies the attracter characteristics.

ATTO (ATTRACT ONE SPECIFIC)

This command sets the action point to attract only a specific action point. If any other moved action point collides it doesn’t attracts.

Point ID	ATTO	PointWaited	ShowMode	NextState	Audio	Overplay

Example: 4 ATTO 3 ONLY_BALL 7 Audio/congratAudio.mp3

Proximity sensor behavior: If actuator it´s not moving a point the sensor doesn´t change, if it´s moving it shows that can be attracted if it´s the waited point, with a CANWORK model. If it´s not the waited point it shows the CANNOTWORK model.

Audio reason: If actuator is moving and collides the waited point plays the audio.

ATTRP (ATTRACT ONE SPECIFIC AND REPELS OTHERS)

This command sets the action point to attract only a specific action point. If any other moving action point collides it repels to the start position.

Point ID	ATTRP	PointWaited	ShowMode	NextState	Audio	Overplay

Example: 5 ATTRP 2 ONLY_BALL 20

Proximity sensor behavior: If actuator it´s not moving a point the sensor doesn´t change, if it´s moving it shows that can be attracted if it´s the waited point, with a CANWORK model. If it´s not the waited point it shows the CANNOTWORK model.

Audio reason: If actuator is moving and collides the waited point plays the audio, otherwise it plays the base error sound.

ATTA (ATTRACT ALL)

This command sets the action point to attract any action point that collides with it.

Point ID	ATTA	ShowMode	NextState	Audio	Overplay

Example: 2 ATTA FLASH_BALL 8

Proximity sensor behavior: If the actuator is moving a point it shows that can attract with the CANWORK model.

Audio reason: Plays on collision with any moving point.

REPEL COMMANDS

Repel is an away to release the point of its moving. When an action point is moved by the actuator and it collides to an action point that is set to repel, it releases the point and send it to the moving action point start position.

RPLO (REPELS ONE SPECIFIC)

This command sets the action point to repel only a specific action point. If any other moving action point collides it doesn’t repels.

Point ID	RPLO	PointWaited	ShowMode	NextState	Audio	Overplay

Example: 3 RPLO 4 BOTH 9

Proximity sensor behavior: If actuator it´s not moving a point the sensor doesn´t change, if it´s moving it shows that can be repelled if it´s the waited point, with a CANWORK model. If it´s not the waited point it shows the CANNOTWORK model.

Audio reason: If actuator is moving and collides the waited point plays the audio.

RPLA (REPELS ALL)

This command sets the action point to repel any action point that collides with it.

Point ID	RPLA	ShowMode	NextState	Audio	Overplay

Example: 2 RPLA FLASH_BALL 8

Proximity sensor behavior: If the actuator is moving a point it shows that can repel with the CANWORK model.

Audio reason: Plays on collision with any moving point.

DROP COMMANDS

The drop is an away to release the point of its moving. When an action point is moved by the actuator and it collides to an action point that is set to drop, the user chooses to release or not the action point, by taking out the actuator.

DRPO (DROP ONE SPECIFIC)

This commands sets the action point to receive a drop request of a specific action point that is been moved by the actuator. If the action point is the waited it releases the point, otherwise repels to the start position.

Point ID	DRPO	PointWaited	ShowMode	NextState	Audio	Overplay

Example: 1 DRPO 5 SENSE_PROX 35

Proximity sensor behavior: If actuator it´s not moving a point the sensor doesn´t change, if it´s moving it shows that can be dropped if it´s the waited point, with a CANWORK model. If it´s not the waited point it shows the CANNOTWORK model.

Audio reason: If actuator is moving and tries to drop the waited point plays the audio, otherwise it plays the base error sound.

DRPA (DROP ALL)

This command sets the action point to receive a drop request of any action point that the user tries to drop.

Point ID	DRPA	ShowMode	NextState	Audio	Overplay

Example: 5 DRPA SENSE_PROX 3 Audio/success.mp3

Proximity sensor behavior: If the actuator is moving a point it shows that can drop with the CANWORK model.

Audio reason: Plays on collision with any moving point.

CONFIGURATION COMMANDS

The configuration commands setup the action point positioning, orientation and size, the position handler structures, the next state and the active object.

CHGST (CHANGE STATE)

This command sets the action point to change to another state if the actuator collides with it.

Point ID	CHGST	ShowMode	NextState	Audio	Overplay

Example: 1 CHGST ONLY_BALL 4

Proximity sensor behavior: Shows that can be touched. Uses CANWORK model.

Audio reason: Plays on collision.

TRA (TRANSLATE)

This command performs an incremental translation (in millimeters) of the actual position of the action point by the three axis. An option to this command is to create a translation animation. When you add the time and step to this command you create a smooth transition from the actual position to the requested.

Point ID	TRA	X	Y	Z	Time	Step

Example: 2 TRA 100 0 0

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

ROT (ROTATE)

This command performs an incremental rotation (in degrees) of the action point actual orientation by the three axis. An option to this command is to create a rotation animation. When you add the time and step to this command you create a smooth transition from the actual rotation to the requested value.

Point ID	ROT	X	Y	Z	Time	Step

Example: 2 ROT 90 0 0

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

SCL (SCALE)

This command performs an incremental scale of the action point model size by the three axis. An option to this command is to create a scale animation. When you add the time and step to this command you create a smooth transition from the actual scale to the requested value.

Point ID	SCL	X	Y	Z	Time	Step

Example 1 SCL 0.5 0.5 0.5

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

CHGM (CHANGE ACTIVE OBJECT)

This command performs a change of the active object from a list of possible objects of the action point.

Point ID	CHGM	ModelToChange

Example: 1 CHGM 4

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

CHGNM (CHANGE TO NEXT MODEL)

This command performs a change of the active object to the next from a list of possible objects of the action point.

Point ID	CHGNM

Example: 3 CHGNM

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

SETS (SET START)

This command (SETS) copies the actual point position to the start point position.

Point ID	SETS

Example: 1 SETS

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

SETL (SET LAST)

This command (SETL) copies the actual point position to the last point position.

Point ID	SETL

Example: 1 SETL

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

GETS (GET START)

This command (GETS) copies the start point position to the actual point position; it changes the point to the start position.

Point ID	GETS

Example: 1 GETS

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

GETL (GET LAST)

This command (GETL) copies the last point position to the actual point position; it changes the point to the last position before the movement. If it hasn't change it assumes the start position.

Point ID	GETL

Example: 1 GETL

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

STATIC COMMAND

The static command (STAT) sets the action point to do not react with another action point or the actuator.

Point ID	STAT	ShowMode	Audio	Overplay

Example: 1 STAT ONLY_MODEL Audio/explosion.mp3

Proximity sensor behavior: Shows that can be touched. Uses CANWORK model.

Audio reason: Plays on collision.

EXTERNAL COMMANDS

These commands sent and receive information from an external source that can be hardware or another program to control basAR.

ESND (ENABLE SEND)

This command (ESND) sends a message that is translated to a command to a extern points, as hardware.

Point ID	ESND	MSG	Audio	Overplay

Example: 1 ESND enableLight

Proximity sensor behavior: Don´t work on external commands.

Audio reason: Plays audio if it´s correctly sent, otherwise plays base error sound.

ERCV (ENABLE RECEIVE)

This command (ERCV) enables the basAR to receive a message from an extern source, as hardware.

Point ID	ERCV	MSG	Audio	Overplay

Example: 1 ERCV buttonClicked

Proximity sensor behavior: Don´t work on external commands.

Audio reason: Plays audio if it´s correctly read, otherwise plays base error sound.

MATH COMMANDS

basAR provides two variables by point to help user to produce some IA or better behavior testing, so, each point has a variable A and B with a double type (+/- 1.7e +/- 308 (~15 digits)). Must be enough to any genius IA.

LOADA (LOAD VARIABLE A)

This command loads the variable A with a double value.

A <= Value

Point ID	LOADA	Value

Example: 1 LOADA 15.0

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

LOADB (LOAD VARIABLE B)

This command loads the variable B with a double value.

B <= Value

Point ID	LOADB	Value

Example: 3 LOADB 15.0

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

ADDA (ADD A VALUE TO A)

This command adds to the variable A, a double value.

A <= A + Value

Point ID	ADDA	Value

Example: 1 ADDA 15.0

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

ADDB (ADD A VALUE TO B)

This command adds to the variable B, a double value.

B <= B + Value

Point ID	ADDB	Value

Example: 3 ADDB 15.0

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

AMB (A MINUS B)

This command subtracts from A the value in variable B.

A <= A - B

Point ID	AMB

Example: 3 AMB

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

BMA (B MINUS A)

This command subtracts from B the value in variable A.

B <= B - A

Point ID	BMA

Example: 3 BMA

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

NEGA (NEGATIVE A)

This command inverts the signal of the variable A.

A <= -A

Point ID	NEGA

Example: 3 NEGA

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

NEGB (NEGATIVE B)

This command inverts the signal of the variable B.

B <= -B

Point ID	NEGB

Example: 3 NEGB

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

MULA (MULTIPLY A VALUE TO A)

This command multiplies the variable A by a value.

A <= A * Value

Point ID	MULA	Value

Example: 3 MULA 5.0

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

MULB (MULTIPLY A VALUE TO B)

This command multiplies the variable B by a value

B <= B*Value

Point ID	MULB	Value

Example: 3 MULB 3.4

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

SWAB (SWAP VALUES OF A AND B)

This command swaps the values from A and B.

A <= B and B <= A

Point ID	SWAB

Example: 3 SWAB

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

CMP (COMPARE A WITH B)

This command tests the variable A with the variable B, if the comparison is correct it changes the state.

The possible tests types are: GREATER / LESSER / EQUALS.

Point ID	CMP	TYPE	NextState

Example: 3 CMP GREATER 4

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

CMPV (COMPARE A WITH VALUE)

This command tests the variable A with a Value, if the comparison is correct it changes the state.

The possible tests types are: GREATER / LESSER / EQUALS

Point ID	CMPV	Value	TYPE	NextState

Example: 3 CMPV 800.0 LESSER 15

Proximity sensor behavior: Don´t work on configuration commands.

Audio reason: Don´t use audio.

STATE MACHINE

State, on the basAR, is a group of actions and configurations that set the behavior at a moment of the application.

basAR provides the possibility to change the states, changing the behavior of the action points. The use of several states allows a more flexible and dynamic application. The link among the states is conceptually equal to the States Machines, proposed by Moore or Mealy, where the choice to go to a state or another is done by a condition.

The action commands have a *NextState* field; this field indicates which state will be loaded if the behavior of that point is achieved.

basAR also provides two types of state: Working state and configuration state.

- The working state waits for a behavior to complete to change to the next state.

- The configuration state reads and applies all behaviors and configurations through the action points and automatically goes to a next state.

The difference between the working state and the configuration state is on the END_STATE statement of the behavior configuration file.

- The working state only uses END_STATE.
- The configuration state sets the next state to go to using: END_STATE **GO_TO <NextState>**.

On the configuration state, it´s also possible to use the AFTER <Time> to delay the beginning of the next state. The end state sentence will be: END_STATE **GO_TO <NextState> AFTER <Time>.**

It’s not necessary to setup all the points every time, as if the state doesn’t change the behavior it keep the behavior of the last state. A Configuration Command doesn’t change the last working command so the behavior it’s also kept after using Configuration Commands.

It’s important that on the first state, in order to configure all the action points, setup all the action points as the application can crash due to unknown behavior.

This state machine can be used to created several types of applications, as puzzles, procedure guides, games, educational activities, etc…

3DMODEL LIBRARY

The library to handle 3D Model on this version is the openVRML. It handles static models, animated models and with texture (.gif). The file must be saved as .wrl, uncompressed.

The file has to be encapsulated on .dat file to be loaded by the openVRML. The .dat file format is on section “Configure an openVRML object file: object.dat” on chapter about Configuration Files.

AUDIO LIBRARY

On basAR it’s possible to use sound on interactions. It’s used irrKlang Library, and the following formats can be used:

- RIFF WAVE (*.wav)
- Ogg Vorbis (*.ogg)
- MPEG-1 Audio Layer 3 (*.mp3)
- Free Lossless Audio Codec (*.flac)
- Amiga Modules (*.mod)
- Impulse Tracker (*.it)
- Scream Tracker 3 (*.s3d)
- Fast Tracker 2 (*.xm)

EXTERNAL HARDWARE CROSS-REALITY LINK

Cross-Reality, on a general sense, is the integration of virtual reality and real reality, where the virtual interaction and the real interaction, with sensors and actuators, can provokes bidirectional changes on both realities.

basAR is Cross-Reality ready, in the fact that allows an extern hardware connection. Commands can be sent and received, allowing the exchange of information between the realities.

The Figure 14 shows the concept of the connection between the real world and the virtual world. Here is shown an ARDUINO as a hardware controller, however could be used any hardware that creates a COM.

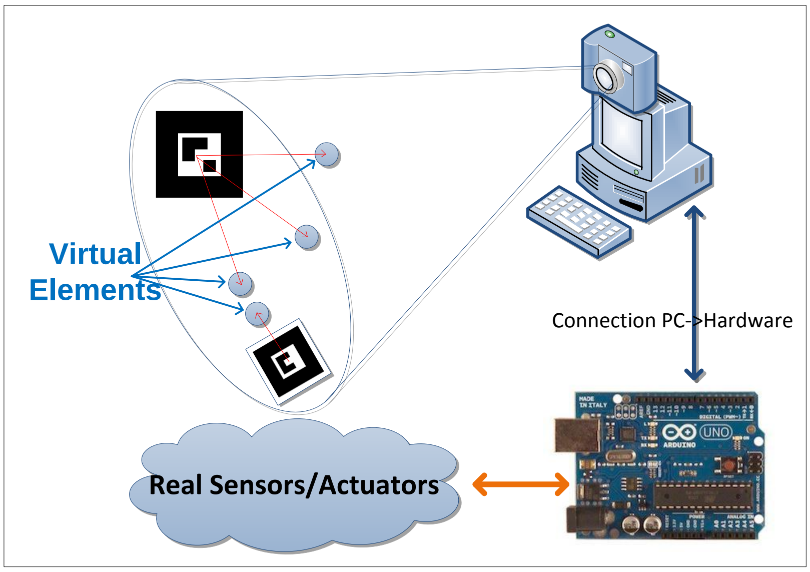

Figure 14 - Cross-Reality concept

An example of cross-reality could be the control of a real and virtual light, from a real and virtual switch, where both switches control both lights. As its shown on Figure 15.

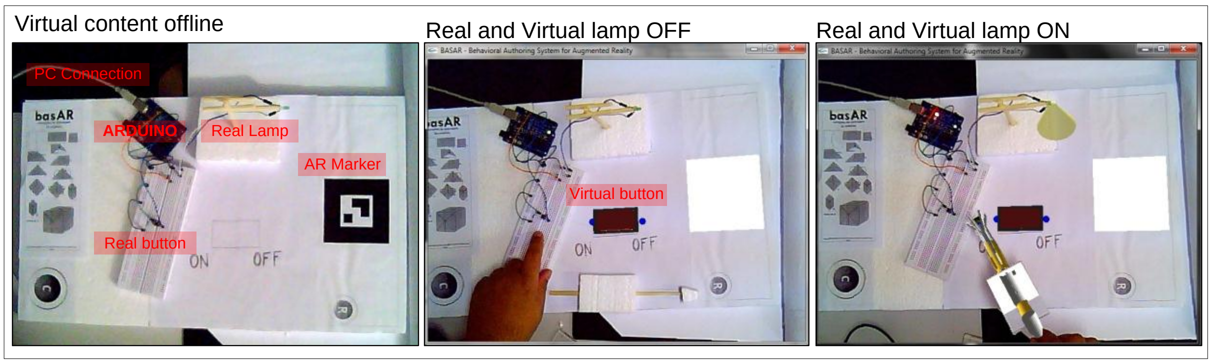

Figure 15

Figure 16 - Cross-reality example

In order to understand basAR Cross-Reality it´s necessary to look up the three actions time lines, a sending command event, a reading request event and an interrupt request event.

On Figure X the sending command event timeline, described as the following steps:

1. The basAR reads the ESND command
2. The basAR verifies if the hardware is connected
3. The basAR searches on the hardware lookup table, in order to find the command to send.
4. If the command is successfully found it is sent to the hardware.

Note that the external sending command doesn´t change state on success, if its desired you may force on hardware an interruption process to change the state.

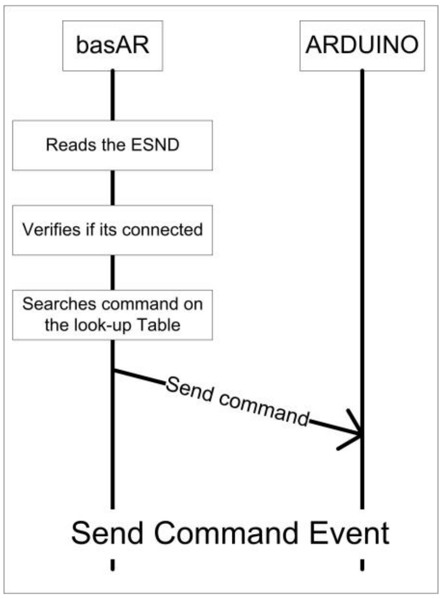
basAR
ARDUINO
Reads the ESND
Verifies if its connected
Searches command on the look-up Table
Send command
Send Command Event

Figure 17

On Figure X the reading request command event timeline, described as the following steps

1. The basAR reads the ERCV command.
2. The basAR verifies if the hardware is connected
3. If the ERCV Command is complete it starts the reading request, otherwise it stops the reading request. (This is used to stop the reading request)
4. The basAR sends to the hardware a readRequest command.
5. Until this command isn´t stopped, it repeats the step 4.
6. The hardware identifies the readRequest and sends to basAR an answer.
7. The basAR searches on the hardware lookup table, in order to find the requested command to send.
8. If the command is successfully found it is sent to the hardware.
9. The hardware reads the command sent and search for a respective answer.
10. The hardware sends to the basAR the command back if it is correct.
11. The basAR reads the answer and test again if it's the requested command, if its correct changes the state.

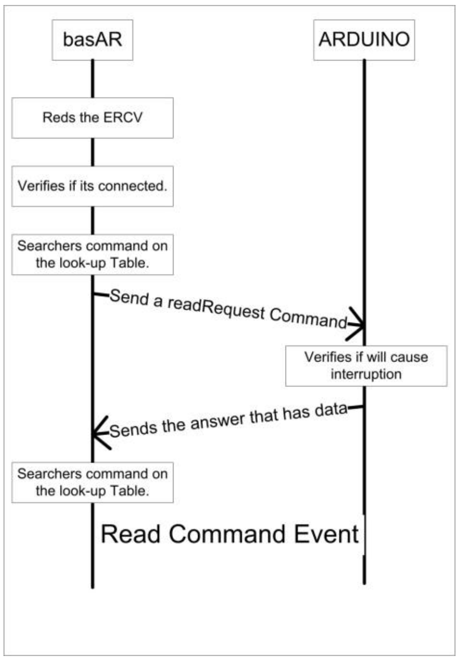
basAR
ARDUINO
Reds the ERCV
Verifies if its connected.
Searchers command on the look-up Table.
Send a readRequest Command
Verifies if will cause interruption
Sends the answer that has data
Searchers command on the look-up Table.
Read Command Event

Figure 18

On Figure X the interrupt request timeline is a loop of tests by the basAR, asking the hardware if it has an interruption to make. This follows a master-slave design. The interruption request works as the following steps:

1. The basAR searches for the intRequest command code.
2. If command is found, it is sent.
3. The hardware reads the command sent and search for a respective answer.

4. The haddware sends to the basAR the new state.
5. The basAR reads the answer and test if the new state is valid.
6. If the state is valid it is changed.

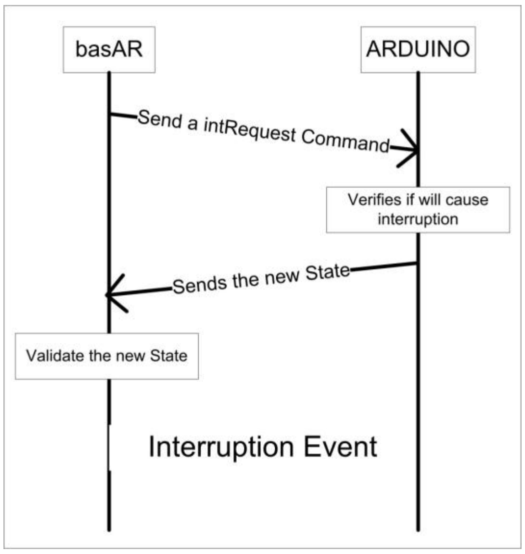

Figure 19

On basAR, the lookup table is as seen on Figure X:

```
COM4 # ARDUINO COM

# ARDUINO LOOKUP TABLE
# THE NUMBERS 00-09, and 255 are RESERVED to Configuration and Test.  10-254
# TABLE FORMAT:   REQUEST_NUMBER REQUEST_NAME <NEXT_STATE>

# CONFIGURATION AND TEST COMMANDS (DO NOT CHANGE THESE REQUEST_NAMEs, ONLY THE REQUEST_NUMBERs)

1 aliveTest
2 aliveAnswer
3 intRequest
4 readRequest
5 readRequestAnswer

# USER COMMANDS

# DEVICE 1 - Light
10        lightOFF
11        lightON

# DEVICE 2 - Button

20        buttonPressed    5
```

The ARDUINO Code example is seen on Figure X:

```
int ledPin = 12;   // select the pin for the LED
int boardled = 13;   // select the pin for the LED
int buttonPin = 2;

int buttonState = 0;
int ledState = 0
int b1Hold = 0;
int val = 0;       // variable to store the data from the serial port
int intNSdata = 0;


void setup() {
  Serial.begin(9600);        // connect to the serial port
}

void loop () {
  // read the serial port
  val = Serial.read();
  buttonState = digitalRead(buttonPin);

                … // Code to adapt interruption


  switch(val){
  case 1: // CHECK IF ITS aliveTest
    {
      Serial.print(2);        // send back aliveAnswer
      break;
    }
  case 3: // CHECK IF ITS intRequest
    { // CHECK IF HAS NMI
      // IF IT HAS NOTHING SEND BACK 0
      // OTHERWHISE SEND THE STATE TO CHANGE THE STRUCTURE STATE MACHINE
      Serial.print(intNSdata);
      break;
    }
  case 4: // CHECK IF Its readRequest
    {
      break;
    }
  default:
    break;
  }

}
```

KEYBOARD OPTIONS

SCREEN MODE

On basAR it is possible three screens mode to adjust to your project: Windowed, full screen and projection.

Windowed

On Windowed the application starts as a windows screen, it´s set

Full screen

Projection

ADDEND 1 – BEHAVIOR COMMANDS SET

Static, no action defined.
[PointID] **STAT** [ShowMode] <AUDIO> <OVER?>
This action allows dragging the action point freely.
[PointID] **DRGF** [ShowMode] <AUDIO> <OVER?>
This action allows dragging the action point freely and returning to origin if collided.
[PointID] **DRGRP** [ShowMode] [NextState] <AUDIO> <OVER?>
This action attracts one specific transporting action point.
[PointID] **ATTO** [PointWaited] [ShowMode] [NextState] <AUDIO> <OVER?>
This action attracts one specific transporting action point and repels others.
[PointID] **ATTRP** [PointWaited] [ShowMode] [NextState] <AUDIO> <OVER?>
This action attracts all action points.
[PointID] **ATTA** [ShowMode] [NextState] <AUDIO> <OVER?>
This action allows dropping one specific transporting action point.
[PointID] **DRPO** [PointWaited] [ShowMode] [NextState] <AUDIO> <OVER?>
This action allows dropping all action points.
[PointID] **DRPA** [ShowMode] [NextState] <AUDIO> <OVER?>
This action repels one specific transporting action point.
[PointID] **RPLO** [PointWaited] [ShowMode] [NextState] <AUDIO> <OVER?>
This action repels all action points.
[PointID] **RPLA** [ShowMode] [NextState] <AUDIO> <OVER?>
This action changes the actual state to a next state.
[PointID] **CHGST** [ShowMode] [NextState] <AUDIO> <OVER?>
This action translates the action point.
[PointID] **TRA** [X] [Y] [Z] <Time> <Step>
This action rotates the action point.
[PointID] **ROT** [X] [Y] [Z] <Time> <Step>
This action scales the action point.
[PointID] **SCL** [X] [Y] [Z] <Time> <Step>
This action changes the active model of the action point.
[PointID] **CHGM** [ModelToChange]
This action changes the active model to the next model of the list.
[PointID] **CHGNM**
This action saves Actual position on Start.
[PointID] **SETS**
This action saves Actual position on Last.
[PointID] **SETL**
This action saves Start position on Actual.
[PointID] **GETS**
This action saves Last position on Actual.
[PointID] **GETL**
This action sends an External command from the lookup table.
[PointID] **ESND** [MSG] <AUDIO> <OVER?>
This action activates a specific listening command from Extern hardware. 0 or ""disables receive.
[PointID] **ERCV** [MSG] [NextState] <AUDIO> <OVER?>
Load variable A with a value. A <= Value
[PointID] **LOADA** [Value]
Load variable B with a value. B <= Value
[PointID] **LOADB** [Value]

Add variable A with a value. A <= A + Value
[PointID] **ADDA** [Value]
Add variable B with a value. B <= B + Value
[PointID] **ADDB** [Value]
Decrement B from A. A <= A – B
[PointID] **AMB**
Decrement A from B. B <= B - A
[PointID] **BMA**
Invert sign of A. A <= -A
[PointID] **NEGA**
Invert sign of B. B <= -B
[PointID] **NEGB**
*Multiply A with a value. A <= A*Value*
[PointID] **MULA** [Value]
*Multiply B with a Value. B <= B*Value*
[PointID] **MULB** [Value]
Compare A with B. If A is (Greater, lesser or equals) it setups a next state.
[PointID] **CMP** [GREATER/LESSER/EQUALS] [NEXTSTATE]
Compares A with Value. If A is (Greater, lesser or equals) it setups a next state.
[PointID] **CMPV** [Value] [GREATER/LESSER/EQUALS] [NEXTSTATE]

[ShowMode] - Point visibility mode
[PointWaited] - Object expect to the action
[NextState] - Next state if action is accomplished
[ModelToChange] - Model to change on CHGM command
[X], [Y], [Z] – Angle coordinates to
<AUDIO> - Sound to play if action is accomplished (May not be used). DRGRP, ATTO, ATTRP, DRPO generate error sound.
<OVER?> - Stops any other sound and play only this one.

Point visibility mode:

HIDE	Hide active model and action point
ONLY_BALL	Show only the action point
ONLY_OBJECT	Show only the active model
BOTH	Show active model and action point
FLASH_BALL	Show only a flashing action point
SENSE_PROX	Shows only the action point with proximity sensor function. • ATTA and DRPA, (Away and correct) • RPLA, (Away and Wrong) • ATTO, ATTRP, DRPO and RPLO (Away, correct and wrong)
ALL_OBJECTS	Show all models and action point

Behavior state:

Normal State	Configuration State	Configuration State with Time
BEGIN_STATE [Number] Actions END_STATE	BEGIN_STATE [Number] Actions END_STATE GO_TO [NextState]	BEGIN_STATE [Number] Actions END_STATE GO_TO [NextState] AFTER [Time]