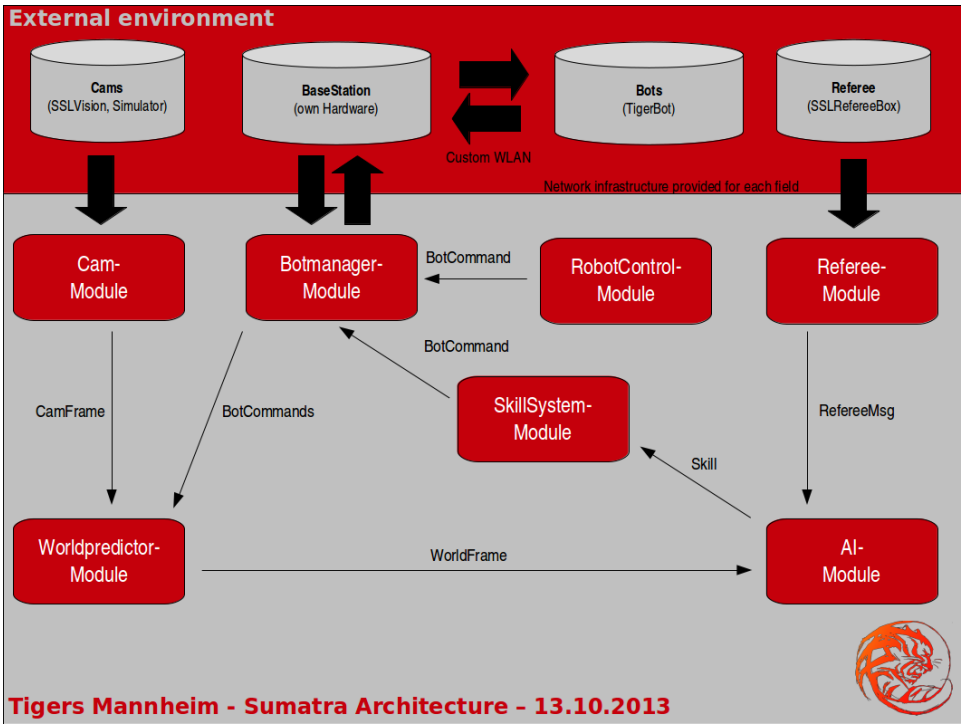
A summary over the AI system

as of: 15.11.2014

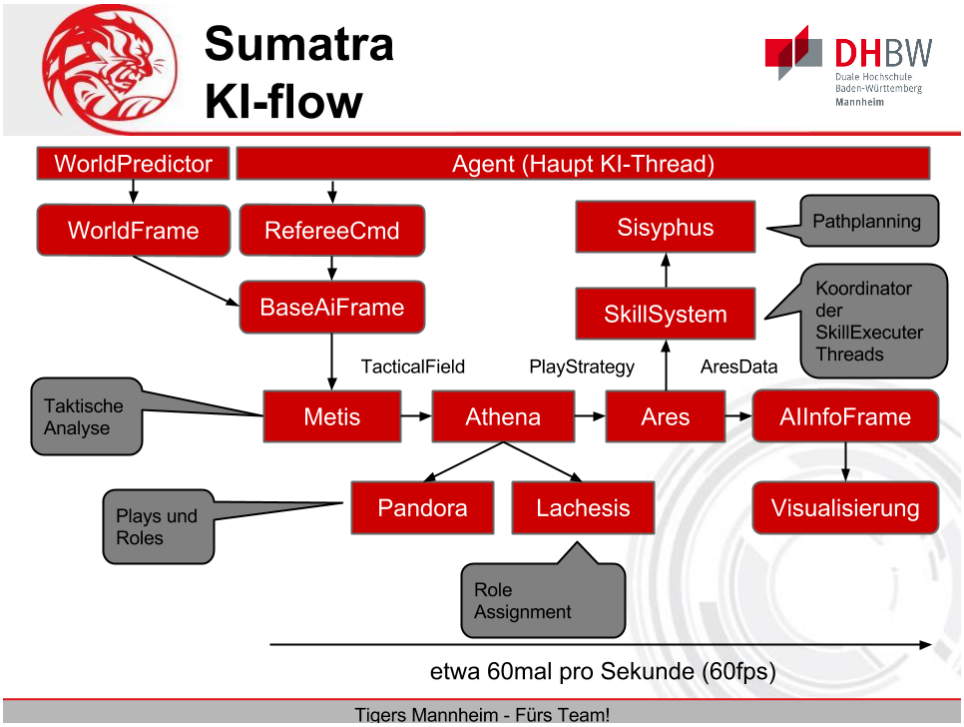
There is also a presentation that you should have a look at:
https://tigers-mannheim.de/owncloud/index.php/apps/files
/ajax/download.php?dir=%2Fpublic%2FDocuments%2Fsonstiges&
files=TIGERS%20-%20Informatik%2BSumatra%20-%20Tigers%20WE%202014.pdf

The big picture

The good old architecture figure:



AI flow



Modules

The AI is divided into several sub-modules as can be seen in the slide above.

If you want to have a look into the code for specific modules, try using Ctrl+Shift+T to find classes, like Metis and Athena or APlay, ARole,ASkill, ACalculator and so on. The enum EPlay, ERole, ESkillName, ECalculator are also useful, because they contain references to all implementations.

Metis

Pre-calculation of useful information and decisions. Basically all real AI calculations are done here.

Don't fear calculating more than is needed. The idea is also, when calculating everything in each frame, we get less performance peaks and the Java JIT can optimize your code early.

Structure

Metis consists of a list of Calculators that are processed one by one. They are defined in ECalculators. The order in the enum specifies the order of execution. When you need to add a new Calculator, create a class and put it near the others. Then add an entry into the enum and specify your class. Everything else is done for you.

Communication

Calculators get as input a BaseAiFrame, which consists of the WorldFrame, referee message and the previous full AIInfoFrame. They can put any results into the given TacticalField. This will also contain data from Calculators that run before. Before accessing such data, make sure your Calculator really runs after the one that produces the data.

Athena

Athena is responsible for executing Plays and Roles and for Role-assignment.

Plays

Initially, Plays were like a fixed strategy with multiple robots, like doing an indirect shot or a pass. It turned out, that this does not work that well for us, because switching between Plays was too slow.

At the moment, there are basically only for concrete plays used during a match: OffensePlay, SupportPlay, DefensePlay and KeeperPlay. Their task is to manage multiple roles. For example, if the SupportPlay gets two roles assigned, it can create two SupportRoles. The roles will have robots assigned, but the play is free to switch the bots among its roles.

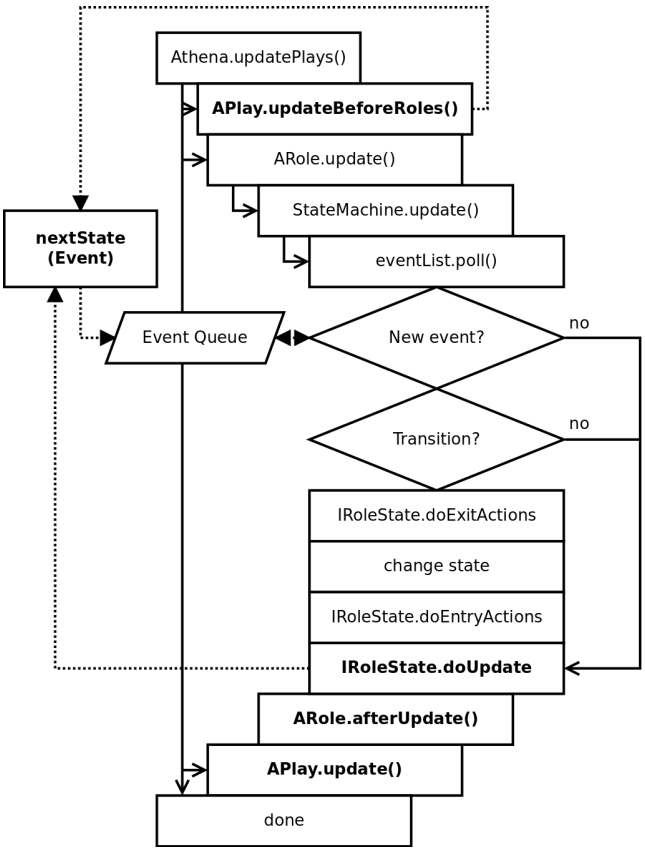
Plays are also used for testing, if multiple roles need coordination.

Finally, there are new plays now that deal with penalty-situations, so that the other plays do not have to deal with this.

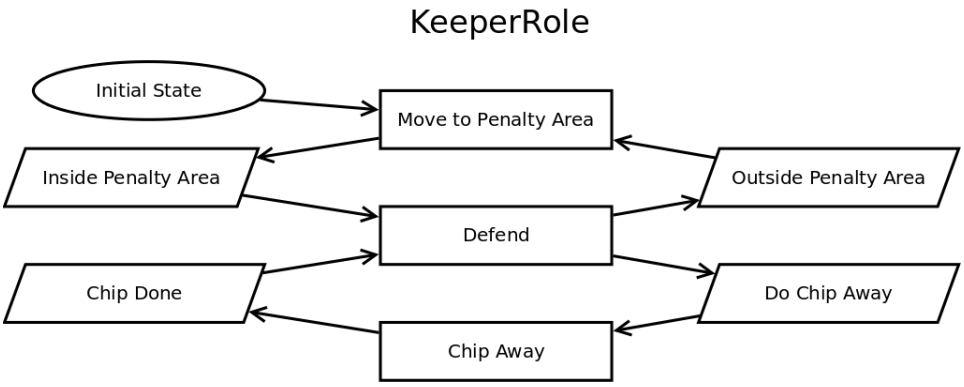
Roles

A role is assigned to a single robot an executes a high level strategy. Ideally, it is not really complex, because it only executes what Metis has calculated.

To make it easier to switch between different states in a role, all roles are managed with a state machine. States are defined as sub-classes. StateIds and Events are defined using arbitrary enums. Transitions are defined in the constructor. They can be triggered with nextState(Event-enum). A transition is not triggered directly, but only on an update of the state machine which is triggered before any role updates. Special care must be taken when generating multiple transitions within one frame (try to avoid it). The following figure shows the complete update cycle:



To give you an idea of how states and a state machine can be useful, here is what the KeeperRole does atm:



The primary goal of a role is to choose the right skill to be executed. The skill can be set with setNewSkill. You should call this on each state change in doEntryActions().

RoleAssigner

The RoleAssigner and its predecessors have a long history.

Initially, we had a PlayFinder that decides which plays to execute next, followed by a RoleAssigner that maps robots to the roles of the plays based on distance.

The PlayFinder became much more complex when we tried to learn a good set of plays automatically, but after 2013 we decided to restructure the AI to be faster and more responsive. This change included removing the original concept of plays and no PlayFinder was needed anymore...

Instead, we got a new RoleFinder. Its task was to decide which bot gets which role. The OffenseRole had the highest priority, and roles were switched quickly when the ball moved. The disadvantage of this RoleFinder was its complexity and some duplicate algorithms.

The next version of the RoleFinder is quite simple again and delegates most decisions. The idea is: Ask each play: How many roles do like? How many could you deal with at most? How many roles do you need at least? And do you have one or more preferred bots? The answers are collected in a Metis Calculator that simply collects the answers from the respective Calculators.

The RoleAssigner will now go through the answers, starting with the play with the highest priority and tries to match the answers. It will do so in each frame, so any changes will usually applied instantly.

Ares

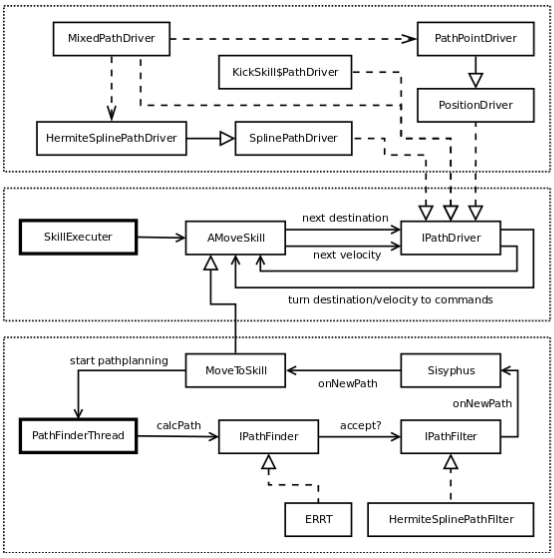
Ares is responsible for skills and pathplanning. It basically executes what Metis and Athena decided.

Skills

A skill defines the movement and behavior of a bot. Each bot can run one skill at a time. A skill can generate a set of bot-commands in each iteration. Common commands are one the one hand a velocity or destination command for movement and on the other hand a command for the kicker and dribbler devices.

You usually won't deal with the commands directly. The class AMoveSkill already helps you with the movement commands and with getDevices() you get good helper functions for kicker and dribbler.

A quite new structure tries to increase flexibility and readability of different implementations within skills for driving. Have a look at the following figure:



Der PathDriver bestimmt das nächste Destination das an den Bot geschickt oder alternativ die nächste Geschwindigkeit.
Einige Implementierungen gibt es schon:
PositionDriver: fährt einfach nur einen Zielpunkt an.
Hier findet allerdings auch die Logik aus dem ehemaligen PositionSkill statt,
die dafür sorgt, dass auch große Entfernungen zurückgelegt werden können.
Außerdem lässt sich die Geschwindigkeit gewissermaßen regeln.
PathPointDriver: Kann einen ganzen Path abfahren, indem er entscheidet, wann welche Node angefahren wird.
SplinePathDriver: Kann einen ISpline abfahren. Ein ISpline gibt eine Position bzw. Geschwindigkeit zu gegebener Zeit zurück.
HermiteSplineDriver: Errechnet aus einem Path einen HermiteSpline und gibt ihn an den SplinePathDriver weiter.
KickSkill#PathDriver: Subclasse des KickSkills. Hier gibt der driver einfach immer den nächsten Anfahrtspunkt zurück, basierend auf der ursprünglichen Berechnung.
MixedPathDriver: Kombination aus 2 IPathDrivern, die abhängig von der Distanz zum Ziel dynamisch ausgewählt werden.

Jeder Skill sollte von AMoveSkill erben und einen PathDriver setzen.
Im AMoveSkill kann eingestellt werden, ob POS oder VEL commands an die Bots rausgehen.

Der MoveToSkill greift direkt auf Sisyphus zu und setzt dort den gewünschten IPathFinder (aktuell nur unser ERRT) und einen IPathFilter (aktuell auf Spline basierter Mechanismus von früher). Beide sind unabhängig von einander. Der Filter hat primär die Aufgabe, nur sinnvolle neue Pfade zu akzeptieren, sekundär aber auch Collision-Detection.
Man könnte den Spline-Filter auch dann verwenden, wenn sonst keinerlei Splines mehr verwendet würden und ein anderer PathFinder im Einsatz wäre.
Der PathFinder wird im MoveToSkill nicht direkt aufgerufen. Das Pathplanning läuft wie früher asynchron in einzelnen PathFinderThreads. Diese werden beim Skill-Start erstellt und gestartet; die Pfade kommen per Observer zurück zum Skill.
So wird der SkillExecuter nicht gebremst, hat aber ggf. ein paar Durchläufe keinen Pfad.

Sisyphus

Sisyphus stands for the never ending task of calculating a good path through all the bots on the field. As there are many fast moving obstacles, calculating a path is quite challenging.

Our path planning implementation is based on ERRT (Extended Rapidly Exploring Random Tree).

The figure above shows that it is possible to also switch to another path finder implementation.

The Pathplanning is executed periodically in its own thread. This is very useful for ERRT, because it is known for its performance peaks that occur due to its random component.

SkillSystem

The SkillSystem manages all the SkillExecuter threads that are created for each bot. Skills are executed independently of the AI.

Last modified on Jul 14, 2015, 5:31:40 PM

Attachments (8)

- TIGERS - Informatik+Sumatra - Tigers WE 2014.png (153.5 KB) - added by *NicolaiO* 8 months ago.
- sumatraArchitecture_20131013.png (151.1 KB) - added by *NicolaiO* 8 months ago.
- Sumatra-Ares-2014.png (114.4 KB) - added by *NicolaiO* 8 months ago.
- RoleUpdateCycle.dia (2.3 KB) - added by *NicolaiO* 8 months ago.
- RoleUpdateCycle.png (87.5 KB) - added by *NicolaiO* 8 months ago.
- StateMachine_Keeper.dia (1.8 KB) - added by *NicolaiO* 8 months ago.
- StateMachine_Keeper.png (38.3 KB) - added by *NicolaiO* 8 months ago.
- Sumatra-Ares-2014.dia (4.6 KB) - added by *NicolaiO* 8 months ago.

Download all attachments as: .zip