The 2005 Report of the IGDA's Artificial Intelligence Interface Standards Committee

Alexander Nareyek, Nathan Combs, Börje F. Karlsson, Syrus Mesdaghi, and Ian Wilson
(eds)

Welcome to our committee's report of 2005, which covers the activities of the committee from June 2004 to June 2005. The first section provides a general introduction and overview while the following sections present details on the committee's single working groups. Feel free to post questions, comments or suggestions to our feedback forum.[1]

Sections:

[1] The current text is a PDF rendering (with minimal formatting) of the latest version of the report available on the Wayback Machine - Internet Archive (https://archive.org/web/), the original report was located at http://www.igda.org/ai/report-2005/report-2005.html

1. Preface

Time goes by at an incredible speed - we already have our third year behind us! The committee was launched on June 17, 2002 within the International Game Developers Association (IGDA). The committee's goals are to provide and promote interfaces for basic AI functionality, enabling code recycling and outsourcing thereby, and freeing programmers from low-level AI programming as to assign more resources to sophisticated AI. Standards in this area may also lay grounds for AI hardware components in the long run.

We had originally planned to be ready with the first interfaces for this year's GDC. However, it's a larger task than we thought, and you need to be a little bit more patient with us. I should be more cautious when handing out ETAs :-)

The number of committee members has increased from 69 last year to 74 at the present time. The committee's members are assigned to working groups, which work on the following topics: World Interfacing, Steering, Pathfinding, Finite State Machines, Rule-based Systems and Goal-oriented Action Planning. As a specific membership category, we have introduced "consultants" now. These "consultants" are expected to review and give feedback on documents that their working group has created, but do not play an active role in the creation of the documents. There is a support team as well, mostly composed of students, who continue to do a great job in supporting the working groups with summaries, documentation and so on. Overviews of the work of all these units during the last year can be found in the following sections.

Again, members' posting in our forums at SourceForge has slowed down to slightly more than 500 messages for the last year. This is partly because working groups like the one on rule-based systems now proceed mostly by an e-mail list, but it is clear that we need to push activity ahead for some groups. This is especially true for the groups on FSM and steering (see this report's section on these groups). On the other hand, groups like the one on goal-oriented action planning have made great progress and are quite close to actual interface specifications.

The new format for our yearly GDC roundtable on AI Interface Standards, where we present our findings and discuss them with the game developer community, worked out very well. Because of the limited time at the roundtable, we focused on a subset of groups this year - those that made the most progress - instead of an overall coverage. Even with this limited set of groups, it was of course too little time to go through and discuss the groups' material in detail, but I think we were able to cover a good amount of issues. You can read more about the GDC session by following the link above.

One thing that we definitely must improve next year is the communication between the groups. For example, as you can read in the world interfacing group's section, some issues there are not compatible with the direction of the rest of the groups.

By the way - please note that we mostly describe only updates on our work in the following report, so feel free to check the ones of the previous years!

Alexander Nareyek
(Committee Chairman)

June 17, 2005
Düsseldorf, Germany

2. General Issues

The committee has a general forum to discuss issues that are relevant for all groups. Some discussions are given below.

2.1. C vs. C++

If you followed previous discussions of the committee, you might remember another discussion on this. While our approach was to have C++ interfaces (including some XML structures for function parameters), C was brought into the discussions again at our GDC roundtable this year. Concerns with C++ interfaces included (non-)compatibilities between compilers, and missing support for C++ on some platforms (e.g., regarding ABIs).

Thus, it was discussed in the committee again. Hardly everyone supported the C++ route, some detailing negative experiences with a C wrapping approaches, and pointing out that new compiler versions, like gcc 4.0, conform to the C++ ABI. If choosing C++, a potential option for vendors that want to also support C interfaces is for example to provide a C library that the C++ library must be linked against.

Considering that our poll at our 2003 GDC roundtable also resulted in a large majority of the participants for C++, the **final** decision was made to **declare C++ as base for the interfaces**. Discussions on what kind of specific language features should or should not get used may of course continue.

2.2. Callback Issues

The question of how to realize callbacks is very important for many parts of the interfaces because middleware often needs to call functions from the game. A unified way of handling callbacks would be an advantage because this makes the application of an SDK/middleware much easier.

There are a number of different ways to implement callbacks, for example:

- **Direct callback functions (like in DirectX)**: The client registers a function with the middleware SDK by passing a function pointer so that the SDK can call the function.
- **Inheritance**: Game objects inherit from the SDK's base objects, and virtual or overridden functions are used as callback mechanism.
- **Message passing**: Some infrastructure is used to exchange messages between the SDK and game engine.

In contrast to direct callback functions, the inheritance approach is type safe and does not require static functions. It is, however, not that easy to "plug in." (An

alternative to direct callback functions might also be the concepts like `Boost.Function`). Message passing is not really an option because we do not want that our interfaces are dependent on specific additional middleware to realize such mechanisms.

For now, the inheritance approach looks most promising, but the discussions have not yet been concluded.

2.3. Call Concept

As an extension of the discussion on "State Handling" of last year's report, here is a brief summary of the discussions. The discussion revolves around the topic of how to structure function calls. These are the main options that were mentioned:

- **HW-F**: hard-wired functions (which was also called *object-oriented*)
  Example:
  ```
  setBlend( true ) or setBlendTrue()
  ```

- **HW-V**: hard-wired enumerations/values (which was also called *state-based*)
  Example:
  ```
  setTrue( BLEND )
  ```

- **DD**: data-driven way
  Example:
  ```
  setTrue( "BLEND" )
  ```

Forward/Backward Compatibility: HW-F obviously has compatibility problems. A new function that is added would lead to an error when called in older versions. In the HW-V approach, passing a non-supported value is not a critical problem. DD obviously also has no problems with compatibility.

Performance: DD is very much in the disadvantage because everything that comes in needs to be interpreted. This is bad for low-level things, but probably fine for higher-level calls that are less often needed.

For now, we will mainly use HW-V because of the compatibility, and DD for some more rare/high-level cases.

2.4. Action Packages / Behaviors

Behavior/action packages was another topic that was discussed. In this concept, behavior components, such as scripts, animations and sounds, are centrally packaged together including conditions/requirements in order to have one central repository for managing potential behaviors/actions. AI sub-systems, like a finite state machine or a planner, can then reference/trigger these behavior objects.

However, the idea did not spark much positive feedback, and also our GDC roundtable participants were quite unsure about the benefits of such a feature. For example, behaviors/actions that need parameters were seen as a potential problem.

3. Working Group on World Interfacing

AI developers and middleware vendors from the game and simulation industries together comprise the AI Interface Standards Committee's (AIISC) world interfacing group. Our groups' mandate is to define 3 basic elements for interactions with the game's virtual world, which are:

a) how AI systems acquire data from the game world ("Sensing"),
b) how AI systems influence the game world ("Acting") and
c) what format the data is sent and received in.

Ian Wilson (Emotion AI, Japan) has the coordinator role and is organizing the groups' work and moderates its forum discussions.

We welcomed two valuable new members to the group this year. Erwin Coumans from Sony Computer Entertainment Europe who brought with him a close connection to the Collada (http://www.collada.org) data format working group being spearheaded by Sony and Marty Poulin from Sony Computer Entertainment America. Both bring valuable insights to the group. Looking at how the Sony Collada effort has progressed (it is a data format initially aimed at standardizing model data formats and has cross industry support from major game developers to major content creation toolkit makers and hardware manufacturers) it is clear that having broad and formal industry support is key to making an initiative like our gain traction. This could inject much needed energy into our efforts and raise the profile of what we are doing as well as giving it further credibility.

While around 30% of the members of the AIISC are from middleware companies we also need to make a stronger connection at the corporate level with these companies to get their buy in and support for what we are doing. They are the key to the whole effort. While we are aimed at game developers the days of game developers building their own tools are numbered, it makes no sense in terms of cost, time, risk or innovation for most developers. For both developers and middleware vendors one of the major hurdles to adoption is integration with the game engine. This is where our efforts could be of greatest benefit to all parties by defining a standard interface that makes the evaluation and integration of any number of AI middleware essentially "plug and play".

Putting the above two aims together would necessitate game AI middleware vendors supporting our data format and API, and this is where wide industry support, greatly increases our chances of acceptance and creates momentum around the whole AIISC effort.

Following is an overview of our requirements and the major directions our group is taking.

3.1. Goals (Requirements)

We would like to define an Application Programming Interface for AI Components that:

1. Supports a large variety of Game World data
    1. Provides a set of standard data types
    2. Supports easily adding new data types
2. Supports a large variety of Game Engine architectures and designs
    1. Supports data transfer between AI and the Game World
    2. Supports AI receiving event data from the Game World
    3. Supports AI making available event data to the Game World
    4. Allows for the future implementation of AI requesting data from Game World
    5. Allows for the future implementation of AI sending data to Game World
3. Supports the API requirements of all groups in the Committee
    1. Supports the "Rule-based Systems" group
    2. Supports the "Pathfinding" group
    3. Supports the "Steering" group
    4. Supports the "Goal-oriented Action Planning" group
    5. Supports the "Finite State Machines" group
4. Supports all committee wide agreed upon formats
    1. Provides documentation in format readable by a web browser
    2. Provides a High Level Specification document of the API
    3. Provides a Low Level Specification document of the API
5. Supports the adoption and use of AI within the games industry
    1. Supports the reuse of API code, allowing developers to avoid "reinventing the wheel" with each product.
    2. Supports reduced development cycles, allowing developers to use existing solutions with greater productivity and familiarity.
    3. Supports the development of more robust Games, allowing heavily tested and well designed interfaces to be used easily.
    4. Supports the development of 3rd party (or shared) AI authoring tools, allowing developers to focus on game content with an externalized tool.
    5. Provides a set of standard terminology that can be adopted by the Game Industry, allowing better communication and common understanding between development studios for higher productivity.

3.2. Overview

This list is intended to define those terms used within the Game / Interactive Entertainment industries to describe components of Artificial Intelligence systems and the Game Systems that are connected or related to those components.

The game development industry is often focused on very low level features in order to squeeze every last drop of performance from a system, however there are a large and growing number of platforms for which games are now developed and as such it would be impossible for us to define a low level system that would be applicable to all platforms. To that end our design is at a higher level. This high level design can be used in its original format or adapted if necessary to specific platforms at a lower level of implementation. Our high level definition is done in two formats that represent the same design but show how it can be implemented in different languages. We have an XML based definition and a C++/Java type definition. These represent a general view of where development is currently and which direction it is taking in the future. It is illustrative rather than complete.

Our current specification has 4 central ideas that define the systems API, these are a) a single, standard, extensible data format is used for all data passing, b) passing data and function calls is transacted via a standard "web service" type methodology, c) receiving data is transacted via a standard "web service" type methodology and d) the game engine can discover from the World Interface which services/methods are available/supported and how they should be called.

While we define an XML type data format for data exchange because we are dealing with a "run time" system it is very probable that this format will be a binary format for faster processing.

The web service methodology is, in its most basic form, a simple standard method of sending and receiving synchronous and asynchronous data to and from an unspecified caller. This is exactly what we would like our interface to accomplish so that it does not require any knowledge of the game world or game engine ahead of time (which we cannot know) but can operate with any engine or world that use the described calling methods and data format. This is the primary aim of the World Interface. Wikipedia defines a web service like this (http://en.wikipedia.org/wiki/Web_service):

A web service is a collection of protocols and standards used for exchanging data between applications. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet, in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and GNU Linux applications) is due to the use of open standards. ...

3.3. World Data Format

The central theme of our data format is that we use a single data format in a standardized way to encompass all data, thus removing the need for a vast array of specific data types for a vast array of possible objects. For a standard system we could not possibly hope to define every possible data type that developers might use and so a standard type becomes essential.

All data fits into a conceptual "attribute/value" pair format at some level whether it is "temperature: 35", "price: 350" or "speed: 160" etc. Attributes can be nested in a hierarchy to group similar elements. Using an XML schema methodology this data format can grow or shrink depending on the requirements of the system, as long as it conforms to the schema. Defining the schema then is the main task of the group. Again, while we talk in XML terms this methodology is a design and as such independent of platform or language.

The general format of attribute/value pairs is simply this:

```
    <attribute>
        <value> </value>
    </attribute>
```

A more concrete example, including a hierarchy, might look something like this (where “element” is the parent node):

```
<elements>
    <scenario>
        <agents>
            <agent id=”Akira”>
                <mood> +23 </mood>
                <energy> tired </energy>
            </agent>
            <agent id=”Yumiko”>
                <mood> +44 </mood>
                <energy> full </energy>
            </agent>
        </agents>
        <objects>
            <chair> wooden01 </chair>
            <chair> leather01 </chair>
            <table> steel03 </table>
        </objects>
        <location> Apartment 101 </location>
        <time> 2135 </time>
    </scenario>
</elements>
```

There are a number of model data exchange formats that have been defined to date and we will use these as a guide and if possible, extend them so we can gain momentum from their community and adoption. So far, these formats have not included other game elements such as events, object meta data (i.e. What an item is as opposed to how it is represented in the game world), relationships etc. These are the elements that, in addition to model data, are required by the AI sub systems. From a developers' perspective they can create their own schema that conforms to the general format but is specific to their own content and is used and validated by their own team for in game data.

3.4. API

The first version of the World Interface Standard will be entirely composed of passive SENSOR, CONTROLLER, and ACTUATOR requests, forming a one-sided API where all function calls are defined AI components only. The controller component is an addition to the previous specification. These 3 terms are often used in game development (in a SENSE - CONTROL - ACT cycle) but simply mean the following:

A method for the Game Engine to pass world data to the World Interface (sensor).
A method for the Game Engine to call functions in the AI sub systems through the World Interface (controller).
A method for the Game Engine to receive data from the World Interface (actuator).
Note there is no method for the AI sub systems to call functions in the Game Engine.

This makes the interface a passive one, receiving data and allowing data to be taken from it. It does not query the world itself or push data back to the world, which would require knowledge of the world and hence a definition of how the world was structured. While it is understood that many developers use query systems it was also felt that moving to an event-based architecture fulfilled our objectives (and the objectives of game developers) in a much more elegant, simple and powerful design.

The "web service" methodology has a number of features but at its core it is also a simple system that defines 4 ways in which a "service" or more specifically an AI sub system can be accessed. These methods depend on whether data is to be returned and whether the call is synchronous or asynchronous (if the calling function is to wait for return data or if return data can follow later). This allows for important calls to wait for important data to return with and also allows non time sensitive calls to simply "drop off" data or trigger actions and then leave the AI system to process the data when it can.

The definitions are as follows:

Type	Definition
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

Notice that "Request-response", which represents the World Interface making a request to the Game Engine is not supported. In essence "Notification" represents the "sensor" sending data to the interface to update the state model, "Solicit-response" represents the "controller" sending a control request and receiving an error response and One-way" represents the "actuator" by receiving data from the interface.

While the actual web service specification is aimed at, well, web services the methodology is equally well suited to any service-based architecture (as our is) where details of one side of the interface are not known ahead of time (again, as ours is). Appendix A has a brief overview of the web service specification and full details are available at the W3C site (http://www.w3.org/2002/ws/) (http://www.oasis-open.org/committees/tc_cat.php?cat=ws).

3.5. API Service Discovery

In order to fully decouple our systems, it is necessary to have in place a mechanism for the Game Engine to query the World Interface to find out what services (AI sub systems and their methods) are available to it and how they should be addressed. This is analogous to the web service UDDI spec (http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm). Again, see Wikipedia's definition here:

UDDI *is an acronym for Universal Description, Discovery, and Integration - A platform-independent, XML-based registry for businesses worldwide to list themselves on the Internet. ...*

UDDI is a one of the core Web Services standards. It is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the web services listed in its directory.

Obviously for our needs communication is generally going to be within a single machine but this method works if the two applications are part of the same larger

application if they are distributed applications or as here entirely separate entities living on the web. Also, the protocols employed for inter application messaging are unlikely to be SOAP but rather function calls. However, we are looking at the bigger picture of the architecture in general and applying it to our needs. From a purely functional perspective this does what we require of a service discovery mechanism. What we would like this discovery mechanism to accomplish are the following:

- *Support querying the interface to find if a module type is available (i.e. path planning)*
- *Support querying the interface to find if a specific module is available (i.e. Path planning / gradient fields algorithm)*
- *Support querying the interface to find what services (public methods) are available, the data they require and the data they return.*

We can, of course, explicitly state what services or methods our interface supports or perhaps create a simple configuration file stating the available options, but these two options are limited and brittle and do not allow our interface to be truly "plug and play". They are, however, simple.

3.6. Group Members

Current members of the working group on world interfacing:

- *Group coordinator:* Ian Wilson - Emotion AI
- Tom Barbalet - Noble Ape
- Axel Buendia - SpirOps
- Erwin Coumans - Sony Computer Entertainment Europe
- John Morrison - MAK Technologies, Inc.
- Gregory Paull - The MOVES Institute
- Borut Pfeifer - Radical Entertainment
- Doug Poston - Conitec
- Marty Poulin - Sony Computer Entertainment America
- Adam Russell - Pariveda
- Duncan Suttles - Magnetar Games

3.6.1. Requirements Sub-Group Members

- Requirement #1 [Data Types]: Ian Wilson, Tom Barbalet
- Requirement #2 [Architecture]: Duncan Suttles, Borut Pfeifer
- Requirement #3 [Data Transfer]: John Morrison, Greg Paull
- Requirement #4 [Other AIISC Groups]: Axel Buendia, Doug Poston

3.7. WG Appendix A: WSDL Documents (Web Service)

A WSDL document is just a simple XML document. It contains set of definitions to define a web service.

3.7.1. The WSDL Document Structure

A WSDL document defines a web service using these major elements:

Element	Defines
<portType>	The operations performed by the web service
<message>	The messages used by the web service
<types>	The data types used by the web service
<binding>	The communication protocols used by the web service

The main structure of a WSDL document looks like this:

```
<definitions>

<types>

   definition of types........

</types>

<message>

   definition of a message....

</message>

<portType>

   definition of a port.......

</portType>

<binding>

   definition of a binding....

</binding>

</definitions>
```

A WSDL document can also contain other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single WSDL document.

WSDL Ports

The **<portType>** element is the most important WSDL element.

It defines a web service, the operations that can be performed, and the messages that are involved. The <portType> element can be compared to a function library (or a module, or a class) in a traditional programming language.

WSDL Messages

The **<message>** element defines the data elements of an operation.

Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

WSDL Types

The **<types>** element defines the data type that are used by the web service.

For maximum platform neutrality, WSDL uses XML Schema syntax to define data types.

WSDL Bindings

The **<binding>** element defines the message format and protocol details for each port.

3.7.2. WSDL Example

This is a simplified fraction of a WSDL document:

```
<message name="getTermRequest">

   <part name="term" type="xs:string"/>

</message>

<message name="getTermResponse">

   <part name="value" type="xs:string"/>

</message>
```

```
<portType name="glossaryTerms">

  <operation name="getTerm">

      <input message="getTermRequest"/>

      <output message="getTermResponse"/>

  </operation>

</portType>
```

In this example the **portType** element defines "glossaryTerms" as the name of a **port**, and "getTerm" as the name of an **operation**.

The "getTerm" operation has an **input message** called "getTermRequest" and an **output message** called "getTermResponse".

The **message** elements define the **parts** of each message and their associated data types.

Compared to traditional programming, glossaryTerms is a function library, where "getTerm" is a function with "getTermRequest" as the input parameter and getTermResponse as the return parameter.

3.7.3. WSDL Ports

A WSDL port describes the interfaces (legal operations) exposed by a web service.

WSDL Ports

The **<portType>** element is the most important WSDL element.

It defines a **web service**, the **operations** that can be performed, and the **messages** that are involved.

The port defines the connection point to a web service. It can be compared to a function library (or a module, or a class) in a traditional programming language. Each operation can be compared to a function in a traditional programming language.

Operation Types

The request-response type is the most common operation type, but WSDL defines four types:

Type	Definition
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

One-Way Operation

A one-way operation example:

```
<message name="newTermValues">

   <part name="term" type="xs:string"/>

   <part name="value" type="xs:string"/>

</message>

<portType name="glossaryTerms">

   <operation name="setTerm">

      <input name="newTerm" message="newTermValues"/>

   </operation>

</portType >
```

In this example the port "glossaryTerms" defines a one-way operation called "setTerm".

The "setTerm" operation allows input of new glossary terms messages using a "newTermValues" message with the input parameters "term" and "value". However, no output is defined for the operation.

Request-Response Operation

A request-response operation example:

```
<message name="getTermRequest">

   <part name="term" type="xs:string"/>

</message>

<message name="getTermResponse">

   <part name="value" type="xs:string"/>

</message>

<portType name="glossaryTerms">

  <operation name="getTerm">

      <input message="getTermRequest"/>

      <output message="getTermResponse"/>

  </operation>

</portType>
```

In this example the port "glossaryTerms" defines a request-response operation called "getTerm".

The "getTerm" operation requires an input message called "getTermRequest" with a parameter called "term", and will return an output message called "getTermResponse" with a parameter called "value".

4. Working Group on Pathfinding

This year most of the effort of the pathfinding group has gone into designing and implementing a generic API that does not dictate a particular world representation or search technique. The group has presented a partial proposal API and accompanying reference implementation that satisfies most past decisions of the group and relevant discussion in other groups. The partial API covers only the graph searching stage of the pathfinding pipeline and not the preprocessing or post-processing stage defined in the 2004 AIISC pathfinding group report.

In order to create a more complete API the group will address the following issues next:

- Handling graph (saving/loading/transferring) in XML format.
- Preprocessing and query input format.
    a. selection between coordinates and node number as query input.
    b. definition of what such a conversion means algorithmically for various graph types.
    c. means to take into account agents.
- Post-processing and path output format.
    a. conversion of the path using node number to a "normal curve".
    b. smoothing/string pulling etc.
    c. API means to select these options, and formats for returning such results from queries.
- Easy interface with the steering API.

4.1. Key API and Design Features

This discussion assumes familiarity with the 2003 and 2004 AIISC pathfinding group reports. We will use the nomenclature (connection, graph, etc.) defined by the group in previous years. The partial API is a single include file defining 3 basic classes:

- An abstract graph object.
- A query object (and one derived object: pathfind query).
- A factory object for creating the former two objects which will be joined with other groups' factories to form a single cohesive library.

The API does not force a specific implementation on the objects so that the underlying implementations can build data structures as they see fit to meet their restrictions (e.g. memory, computation) and optimize. The factory object can cope

with several simultaneous implementations, assuming different library providers can have products with different strengths and weaknesses.

The API allows the library user to specify his own world representation at run-time. Several world representations can be used together through a common interface: The nodes and connections follow a "flexible format" methodology (similar to DirectX vertex format):

- The user defines his C++ structures for nodes and connections: the type of data associated with each node and connection (e.g. position vector for nodes, multiple weights for connections). There is no inheritance of a base node/connection.
- The formats are determined at run-time by passing a macro-constructed descriptor to the API. The macro tells the library how many coordinates there are, how many weights and additional data.
- All subsequent operations and functions follow that format.
- Example: a node can be defined to have 1 vector (for waypoint), 3 vectors (triangular navmesh). The vectors might be 2D or 3D, etc.
- Example: connections can hold several weights (to encode graphs for different units), or height/width (so that only units with lesser dimensions can pass). This way the library user can encode different graphs in the most suitable and convenient way.

The API is state driven in respect to manipulating the state of the pathfinding pipeline to ensure forward compatibility. The API defines some enumeration for selecting the type functionality (e.g. type of heuristic function, selecting between preprocessing and postprocessing in the future). Only a handful of values are currently defined. Other features:

- Queries can be performed in batches. The implementation may change the queries' order for optimization purposes.

4.2. New API

The mechanism for adding a new implementation is of main importance as the API is intended for both users (studios) and library writers (middleware providers), and the API has two parts respectively. One is targeted at users described above and the other at library writers.

The second interface forms a sort of Library Development Kit and provides features for easily dealing with the flexible graph format. Using it also allows interchanging an implementation for a better one without the need to recompile the game or even use several implementations simultaneously (e.g. one library is faster but doesn't implement all the API functions).

4.2.1. Understanding the API Through a Trivial Example

```
// The API
#include "ais.h"

// Define the node and vertex format at run-time
// a 2D waypoint graph with a single weight for each connection
const AISNodeFormat gFNF = AISNODE_POSITION(AISTYPE_FLOAT, 2);
const AISConnFormat gFCF = AISCONN_WEIGHTS(AISTYPE_FLOAT, 1);

struct CNode{
        AISVector2Float v;            // same as float x,y;
};

struct CConnection
{
        UINT32 mNodeFrom;
        UINT32 mNodeTo;
        float mWeight;
};

// Use the factory to create a graph object that will hold all
// these connections and nodes (we use the ref. implementation)
ais_CGraph graph = GetAis()->CreateGraph(AISIMP_GENERIC,
gFNF, gFCF, gMaxNodes, 0);

// Insert nodes and connections into the graph
graph->SetNodes(0, gNumNodes, gNodes);
graph->SetConnections(gNumConnections, gConnections);

// Optimize graph structure to get better performance
graph->Process();

// Perform 2 pathfinding queries on the graph in a batch
ais_CQuery query1 = graph->AddQueryFindPath(StartNode, EndNode);
ais_CQuery query2 = graph->AddQueryFindPath(StartNode, EndNode2);
graph->FlushOperations();

// Get the result of the first query
UINT32 resultSize = graph->GetQueryResults(query1, buffer, true);
```

4.3. Group Members

Current members of the working group on pathfinding:

- *Group coordinator:* Syrus Mesdaghi - Dynamic Animation Systems
- *Co-coordinator:* Noel Stephens - Paradigm Entertainment
- Ramon Axelrod - AiSeek
- Emmet Beeker - MITRE
- Mark Brockington - BioWare Corp.
- Mike Ducker - Lionhead Studios
- John Hancock - Factor 5
- Arno Kamphuis - Utrecht University
- Stephane Maruejouls - MASA Group

- Ian Millington - Mindlathe
- Thomas Young - PathEngine

5. Working Group on Rule-based Systems

In this report we introduce the work carried-out by the Artificial Intelligence Interface Standards Committee (AIISC) working group on Rule-based Systems (RBS). A significant topic of discussion during the 2004-2005 period has been to identify the relationship between Rule-based Systems and in Game Artificial Intelligence. When are Rule-based Systems and rule programming used in Game Artificial Intelligence? How does this relate to scripting, and for what types of games?

Most of the discussion/work conducted in this period is a continuation of previous years. Unique to this year we conducted an in-depth examination of Age of Empires rule scripts (from mods). We also garnered additional insight from discussion at the Game Developers Conference.

Presented here are background concepts and issues discussed on the aspects of specification, design and development of a Rule-based System for game Artificial Intelligence (AI) to implement challenging game agents (i.e., NPCs - "Non-Player Characters"). We are investigating the components and possible architectures of RBS, and the different applications of game AI in general and of RBS in particular, depending on game genres.

5.1. Goals

Rule-based Systems can be used to encode behavioral rules that capture knowledge about a particular game scenario and the agents that inhabit it (e.g., game NPC opponents). In all but the simplest of games, rules that govern the behavior of entities within the virtual world of a game need to embody complex relationships between large numbers of rapidly changing aspects of the overall game state. Programming these behavioral rules is an acknowledged problem in computer game construction.

We expect a trend in the games industry towards more rule-based styles of AI programming. At recent Game Developer Conferences, speakers (e.g., Peter Molyneux and Will Wright) indicated that game AI will play an increasingly important role as a source of "dynamics" and "emergent behavior" that leads to new (generated/emergent) content within games. We believe that this trend will drive game development towards more sustainable programming styles based on "rules." This report introduces the work carried-out by the AIISC's working group on Rule-Based Systems (RBS) to date.

Rule-based Systems are comprised of a database of associated rules. Rules are conditional program statements with consequent actions that are performed if the specified conditions are satisfied.

The aims of the group are to discuss and develop a set of standards on RBS applications and architectures suitable to games. This is a preliminary report on the current work of the RBS group, and which will be followed-up by recommendation and a proposal for a game-AI RBS interface standard in future reports.

- Our interface will assist game developers using an RBS to separate the representation of knowledge and the behavior of intelligent entities in games from their implementation in software.
- Our interface will assist game developers using an RBS to easily integrate game-centric interactions involving the game engine and game entity AI.
- Our interface will provide a rule-structure definition language that is compatible with representation of knowledge and decisions in game systems using the RBS.
- With our interface and our reference implementation we will be able to identify general usage patterns applicable to different game genres.

The effort of this working group will also indirectly assist the game developer community in other ways:

- It will identify knowledge and reasoning patterns for building better and faster game AI using RBS.
- It can suggest means for architecting games better using RBS - more modular design, reusable components, and scriptable rules for more easily customized game AI.
- It can suggest means for architecting RBS that are more compatible with games requirements.

5.2. Major Findings Updated

In the 2003-2004 final report we presented four major findings, representing the conclusions of the group to date. In this year's report we updated them with amendments and new detail.

5.2.1. Increased use of rule-based programming in game Artificial Intelligence expected.

We believe that game AI will rely increasingly upon rule-based and declarative programming techniques in the future. Our expectation is that rules adoption will likely occur first with server-based games (e.g., online multiplayer games), for a couple of reasons. First, server architectures tend to be already largely componentized (e.g., database, login, etc.) - adding a new RBS component on the server is likely more feasible than on highly optimized clients. Second, server-resident games share a number of requirements with commercial server applications, e.g., performance, scalability, and supporting dynamic loading and "hot swapping" of AI logic.

5.2.2. Rule-based Systems can be integrated into games using industry compatible interfaces.

At the level of integration, script interpreters and RBSs pose similar problems. They both have to relate to the game engine. Is their relationship synchronous? Where is the game state? Will they support functional call-backs? Etc. Should the game engine be able to reach-in and tune performance? We have hypothesized how these approaches might ultimately converge behind an interface rooted in an industry standard, **JSR-94** (Java Specification Requests). We believe that a JSR-94-based interface can support a range of RBS middleware (more powerful to less powerful RBS components). We start with the view that an RBS is a rule-engine that (words adapted from JSR-94):

- Acts as an if/then statement interpreter. Statements are rules.
- Promotes declarative programming by externalizing ...game logic.
- Acts upon input objects to produce output objects. Input objects are often referred to as facts and are a representation of the state of the ...game. Output objects can be thought of as conclusions or inferences and are grounded by the game in the... game domain.
- Executes actions directly and affect the game, the input objects, the execution cycle, the rules, or the rule engine.
- Creates output objects or may delegate the interpretation and execution of the output objects to the caller.

From the commercial applications sector we see a trend towards merging rules with scripting (or "rule-based scripting"). This sector has evolved a range of products that uses rule-based scripting to customize middleware. Consider large system architectures servicing business processes. Typically such architectures integrate a range of products and contain much "glue code" (including scripts) whose behavior is conditioned on the values in the data as it passes through, e.g., real-time data feeds etc. Such systems often represent metadata using rules: by separating the representation from the implementation (code) it simplifies maintenance.

While rule-scripting for single-player games can be simple, RBS integration with multi-player online game servers will require considerations beyond a JSR-94-based interface. For example, discussion in games technical forums have from time-to-time suggested that RBS optimizations that work well in commercial business domains may need to be modified for use in online games (e.g., RETE-based rule matching). By first agreeing upon the interface, however, the games industry can then let the middleware providers compete on specific solutions.

5.2.3. The JSR-4 RBS interface is a good foundation, but will likely need to be extended to support game Artificial Intelligence.

Within the RBS working group, we discussed what interface extensions to the JSR-94 will we likely need to support in an RBS API proposal. Specific discussions included:

- Should an API be specified to enable caller to throttle engine resource-usage, e.g., with respect to: memory usage, speed, caching etc.?
- Should an API be specified to support object-based scoping of rules? Objects would correspond to game world entities on the engine.
- Should it be an RBS requirement that an RBS support hot-swapping of rules (particularly in server-based games)
- Should the interface specification be in C++ or Java? The answer to this question is related to whether the first audience of the specification lies with single-player or server-based games.

5.2.4. Caveats with using rule-based systems with game Artificial Intelligence.

Within the RBS working group, we identified a number of issues that developers and middleware implementers will need to consider and address long-term in the game Artificial Intelligence domain:

- **Learning curve.** Declarative programming is as different from object-oriented programming as assembler or SQL. It is not necessarily more or less difficult; it is simply different. Rule-based coding, like OO, is easy to do badly, so project teams should have a fair amount of experience to use rules effectively.
- **Tool support.** Today, few rule authoring systems offer IDEs and debuggers as polished and functional as those available in modern OO languages. Because of the proprietary nature of most rule-based system tools, there are no general development tools as they run on many platforms.
- **Greater control of the execution of the RBS** - e.g., an additional API so that developers can exert greater control over what or how the rule engine is doing. Memory, thread control etc. are examples.
- **Game-centric RBS optimizations.** For example, the widely used RETE algorithm speeds performance by using substantial amounts of RAM. Because of the way the RETE algorithm is designed, there must be a separate rule base loaded for each agent - memory sharing is not possible. The Soar Quake Bot ran on a dedicated machine and that machine could only support 10 bots. Alternative algorithms or use-patterns may need to be identified for game Artificial Intelligence.

5.3. Towards an Interface Specification

This year at the 2005 Game Developer Conference we discussed porting the JSR-94 Java interface into C++. We're waiting upon resolution of the group discussion upon the right use of C vs. C++ within the standard group interface (e.g. maximize portability between console and PC games).

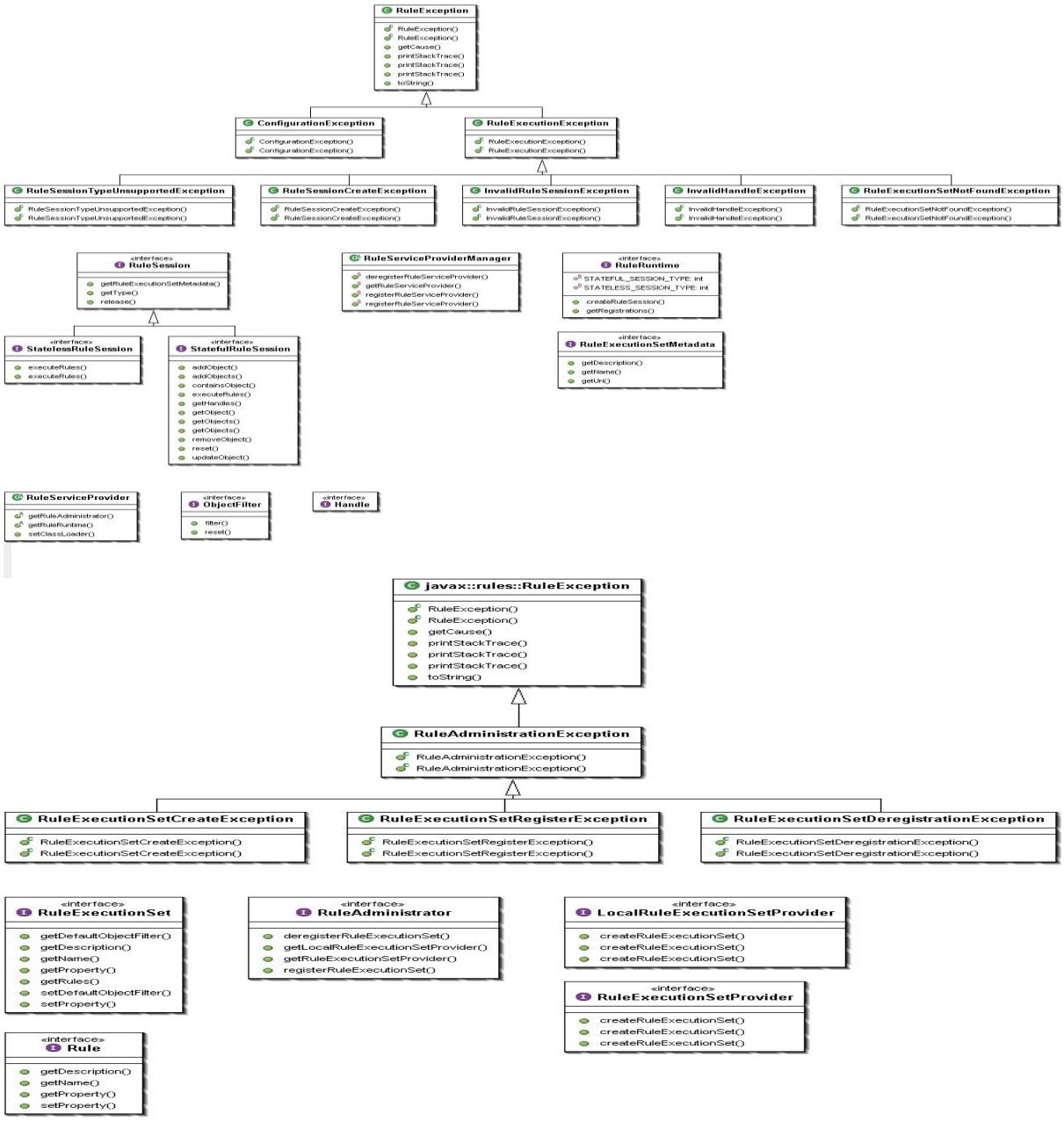
We discussed the minimal demands of the JSR-94 interface. On the one hand this enhances the potential for broad adoption in the industry. On the other hand a thin interface also means that there are likely to be few "game-specific" features in the final specification. However, given the broad range of games, platforms, and rule-uses, we felt that it is better to "keep it stupid simple" and aim for a broader adoption.

We also discussed the differences and preferences of game developers towards "stateless" or "stateful" interactions with RBSs. With stateless interaction the rule engine can be used as a way to implement a function or a service. You call it, passing parameters to it. It returns a result as an object. The rule engine doesn't keep any knowledge of previous calls (statelessness). With a stateful interaction the rule engine permits a client to have a prolonged interaction with a rule engine within a single session. Input objects can be progressively added to the session and output objects can be queried repeatedly. Based on the GDC discussion, we concluded that both interaction models (which JSR-94 supports) should be supported.

To illustrate the simplicity of an AIISC interface based on the JSR-94 pattern, we also discussed a number of minimal examples of how a game engine would interact with the RBS via a JSR-94 interface, e.g. (Note the example below is Java):

```
// Get the rule service provider from RuleServiceProvider manager
RuleServiceProvider serviceProvider =
RuleServiceProviderManager.getRuleServiceProvider(RULE_SERVICE_PROVIDER );

// Obtain runtime instance from provider
RuleRuntime ruleRuntime = serviceProvider.getRuleRuntime();

// Create a StatelessRuleSession from runtime
StatelessRuleSession statelessRuleSession =
(StatelessRuleSession) ruleRuntime.createRuleSession(uri,
new HashMap(), RuleRuntime.STATELESS_SESSION_TYPE);

// Create an input list/ Populate it
List input = new ArrayList(); input.add(inputCustomer);
input.add(inputInvoice); //…

// Execute the rules without a filter.
List results = statelessRuleSession.executeRules(input);

// Obtain results – Iterator interface
Iterator itr = results.iterator();
while(itr.hasNext()) {
      object obj = itr.next(); // Do something
}
```

Again to reinforce the simplicity of the JSR-94 interface, we also discussed (Java) UML class diagrams (below). Note that approximately 1/2 of the classes defined here are Exceptions - emphasizing the simplicity of the core classes. *(click on images for a larger version)*



5.4. RBS Introduction

Rule-based systems differ from standard procedural or object-oriented programs in that there is no clear order in which code executes. Instead, the knowledge of the expert is captured in a set of *rules*, each of which encodes a small piece of the expert's knowledge.

Each rule has a left-hand side and a ride hand side. The left-hand side contains information about certain facts and *objects* which must be true in order for the rule to potentially fire (that is, execute).

Any rules whose left-hand sides match in this manner at a given time are placed on an *agenda*. One of the rules on the agenda is picked (there is no way of predicting which one), and its right-hand side is executed, and then it is removed from the agenda. The agenda is then updated (generally using a special algorithm called the *RETE algorithm*), and a new rule is picked to execute. This continues until there are no more rules on the agenda.

5.4.1. Rule-based Systems Components

Rule-based systems consist of a set of rules, a working memory and an inference engine. The rules encode domain knowledge as simple condition-action pairs. The working memory initially represents the input to the system, but the actions that occur when rules are fired can cause the state of working memory to change. The inference engine must have a conflict resolution strategy to handle cases where more than one rule is eligible to fire.

A rule-based system consists of:

- a set of rules,
- working memory that stores temporary data,
- inference engine.

The inference mechanisms that can be used by inference engines are:

- Backward Chaining:
  - To determine if a decision should be made, work backwards looking for justifications for the decision.
  - Eventually, a decision must be justified by facts.
- Forward Chaining
  - Given some facts, work forward through inference net.
  - Discovers what conclusions can be derived from data.

5.5. Discussion: Game-AI Behaviour as Rules or Not?

One of the most significant problems in NPC behaviour control construction is that procedural programming languages such as C, C++ and Java that are typically used in the implementation of computer games are not suitable for implementing large sets of complex behavioural rules. Attempting to construct complex behavioural rules in a language such as C++ tends to result in an impenetrable code involving large numbers of if-then-else constructs, switch, case and loop statements. The programmer is responsible not only for defining the conditions under which a

behavioural rule is to be applied, and the action to be taken when it does apply, but must also deal with deciding all of the circumstances under which to check the game state against the behavioural rule's pre-conditions. Many of the behavioural rules required for game play will interact with one another in ways which cannot easily be determined a-priori. *A game continuously changes during development; and having an easily adaptable system for changing/adding/removing rules is of significant benefit.*

In general, the result of attempting to construct complex NPC behaviour using procedural programming techniques will be complex code, which is incomprehensible and difficult to maintain, leading to an unavoidable increase in development time and requirements for large memory space and processing power. A Rule-based System enables a more flexible, scalable, robust way to design such behaviour. The reasons are grounded in pragmatics, of scalability, of usability, and of logic expressiveness.

- **Pragmatics**: separates implementation from the logic of the behaviour. This can lead to more maintainable engineering and code.
- **Scalability**: easier to separate logic about type or class from logic pertaining to the instance.
- **Usability**: likely more usable to mod developers - witness this trend with business software. Increasingly games builders are looking to appeal to 4th party developers (see Ben Sawyer) to develop outside content to extend the life of the product as well as to broaden its appeal. Such could lead to a "virtuous cycle" equivalent to one that developed in the applications sector, where we saw tools emerge for use with commercial development of large rule-based systems.
- **Logic Expressiveness**: The expressive need for both declarative and imperative forms is straightforward: sometimes it is just easier to think in rules and compute consequences; sometimes it is vice versa. Consider two different approaches for specifying behaviour: the first approach (imperative) is to describe the consequences or the process first; the second (declarative) is to describe the goals or rules first.

Programming game AI using rules can be more pragmatic than scripting because it separates implementation from the logic of the behaviour. This can lead to more maintainable engineering and code. Furthermore, a rule-based representation can provide a more concise and direct relationship between specification and implementation that would simplify testing. Separating the game engine from the game rules allows independent simulation and testing of each. Separating the logic from the implementation also enables reasoning about the logic by itself:

- Is it complete?
- Are all rules reachable?
- Can we use look-ahead?

- Can learning algorithms be applied?

The earlier sections illustrate some of the possible application for AI in game in general and RBS in particular. These are the creation of interesting opponents, realistic and engaging NPCs and maintaining consistency in dynamic storylines.

To help in making the game immersive and allow suspension of disbelief, the creation of the NPCs must provide different types of behaviour. Two categories of generic behaviour in NPCs are commonly used: reactionary and spontaneous.

NPCs behave in a reactionary manner whenever they are responding to a change in their environment. If an enemy spots you and starts to run towards you and shoot, then they have acted as a reaction to seeing you.

NPCs behave in a spontaneous manner when they perform an action that is not based on any change in their environment. An NPC that decides to move from his standing guard post to a walking sentry around the base has made a spontaneous action.

5.5.1. SOAR-BOT Example

Example: SOAR-Quake, courtesy of J. Laird.


```
IF
    enemy visible and my health is < very-low-health-value (20%)
OR
    his weapon is much better than mine
THEN
    propose retreat
```

5.6. RBS Programming

Rule-Based programming is based on a simple model of computation in which knowledge is encoded in the form of sets of condition-action pairs known as production rules. The condition part of a production rule represents the pattern which must be matched in order for the rule to be applied, and the action part of the production rule represents the response that is to be made when the rule is applied. In a game application, the condition part of the production rule might involve interrogating the current state of the game, using virtual *sensors* and the action part of the rule might involve some response to the game state, effected via virtual *actuators*. Rules might also embody intelligent behavioural knowledge at a much higher level, involving planning actions to enable a game agent to achieve high level strategic goals, for example.

Many rule-based programming languages have been implemented, such as OPS5, CLIPS, Jess or RC++, for use in a large number of intelligent systems applications, including Expert Systems and Intelligent Software Agents.

A program written in a rule-based programming language is executed by using a working memory of assertions about the state of the world within which the program is operating, and an interpreter, which matches rules to the working memory, and adds to and removes assertions in response to the actions undertaken by rules that have been executed. A program expressed as a set of production rules is entirely declarative; production rules embody no procedural knowledge. It is the responsibility of the interpreter to control the execution of the rules. This is one of the principal advantages of rule-based programming. The programmer can concentrate on producing a discrete body of rule logic, separated from other aspects of the system, by writing a set of modular rules that are easily understood, easily extended, and easily modified.

5.6.1. Interface with the Game World

An important aspect of the architecture of the RBS is the interface with the game world and how they should be integrated. The world interface is still at a specification stage. However, the components proposed in the current report, such as events, actions and sensors are the main information required from the RBS to enable the NPCs to have the appropriate behaviour and interact with the world in a suitable way. The details of the integration are of course very dependent on what the final world interface will look like.

5.6.2. Research Extensions to Rule-based Systems

Rule-based systems support formalisms with different level of expressiveness. Examples of these include:

- propositional logic,
- first-order logic,
- events and temporal constraints,
- probability associated with rules,
- fuzzy logic,
- etc.

All of these can be used to provide better AIs, e.g., you can imagine a bot that hides when it is being shot at. Then it waits for five seconds before trying to shoot back if there is no other shooting and no incoming noise.

5.6.3. RETE Algorithm

The RETE Algorithm is widely used in commercial RBS implementations - it is regarded as the most efficient algorithm for optimizing mainstream commercial RBS systems. Because of its importance we introduce it here. However, as we have discussed within the working group, the RETE algorithm may not be optimal for game AI applications. We highlight this particular concern by this discussion, as it has been a recurring topic of discussion over the past year.

The efficiency of the RETE algorithm is asymptotically independent of the number of rules. Although a number of algorithms implementing production rules have been considered, based on actual, empirical evidence, the RETE Algorithm is orders of magnitude faster than all published algorithms with the exception of TREAT algorithm. RETE is usually several times faster than TREAT for small numbers of rules with RETE's performance becoming increasingly dominant as the number of rules increases.

The typical RBS has a fixed set of rules while the knowledge base changes continuously. However, it is an empirical fact that, in most RBSs, much of the knowledge base is also fairly fixed from one rule operation to the next. Although new facts arrive and old ones are removed at all times, the percentage of facts that change per unit time is generally fairly small. For this reason, the obvious implementation for an RBS architecture is very inefficient. The obvious implementation would be to keep a list of the rules and continuously cycle through the list, checking each one's left-hand-side (LHS) against the knowledge base and executing the right-hand-side (RHS) of any rules that apply. This is inefficient because most of the tests made on each cycle will have the same results as on the previous iteration. However, since the knowledge base is stable, most of the tests will be repeated. You might call this the *rules finding facts* approach and its computational complexity is exponential.

A very efficient method known is the RETE algorithm. It became the basis for a whole generation of fast expert system shells: OPS5, its descendant ART, RETE++, CLIPS, JESS, and ILOG-Rules.

5.7. Specification

The aim is to propose a general game AI engine organized around the components mentioned in the RBS definitions section. This should make the implementation of NPCs easier by providing a suitable interface with the game-world, a common inference engine and different knowledge base suitable for a large variety of games.

A summary of RBS components is below with possible choices:

5.7.1. Knowledge Representation

- Production Rules
- Frames
- Object-oriented Representation

5.7.2. Inference Algorithm

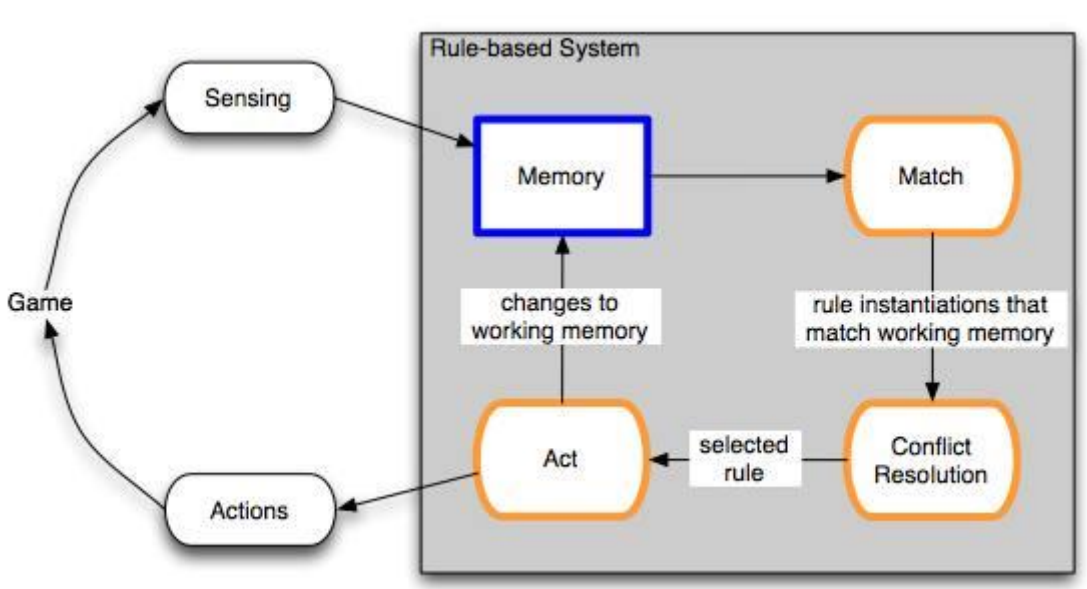
- RETE
- TREAT

5.7.3. System Architecture

- Centralized
- Multi-Agent
- Blackboard

5.7.4. RBS Game AI Cycle

The diagram below shows the "thinking" process.



5.8. Group Members

Current members of the working group on rule-based systems:

- *Group co-coordinator:* Nathan Combs - BBN Technologies
- *Group co-coordinator:* Abdennour El Rhalibi - Liverpool John Moores University
- Jean-Louis Ardoint - ILOG
- Daniele Benegiamo - AI42
- Hannibal Ding - Interserv Information Technique
- Clay Dreslough - Sports Mogul
- Frank Hunter - Adanac Command Studies
- Gerard Lawlor - Kapooki Games
- John Mancine - Human Head Studios
- Miranda Paugh - Magnetar Games
- Baylor Wetzel - GMAC RFC

5.9. WG Appendix A: Concepts and Terminology

Blackboard architecture

A Blackboard Architecture is an AI solution where Knowledge for a domain is shared between numerous KS (Knowledge Sources). Each KS represents an expert bringing its own set of knowledge to the blackboard and uses the knowledge published through the blackboard to build assumptions, make deductions etc.

Condition-action rule

A condition-action rule, also called a production or production rule, is a rule of the form: *if condition then action.*

The condition may be a compound one using connectives like *and*, *or*, and *not*. The action, too, may be compound. The action can affect the value of working memory variables, or take some real-world action, or potentially do other things, including stopping the production system. See also inference engine.

Conflict resolution

Conflict resolution in a forward-chaining inference engine decides which of several rules that could be fired (because their condition part matches the contents of working memory should actually be fired.

Conflict resolution proceeds by sorting the rules into some order, and then using the rule that is first in that particular ordering. There are quite a number of possible orderings that could be used.

Frames

Frames are a knowledge representation technique. They resemble an extended form of record (as in Pascal and Modula-2) or struct (using C terminology) or class (in Java) in that they have a number of slots which are like fields in a record or struct, or variable in a class. Unlike a record/struct/class, it is possible to add slots to a frame dynamically (i.e., while the program is executing) and the contents of the slot need not be a simple value. There may be a demon present to help compute a value for the slot.

Demons in frames differ from methods in a Java class in that a demon is associated with a particular slot, whereas a Java method is not so linked to a particular variable.

Heuristic

A heuristic is a fancy name for a "rule of thumb" - a rule or approach that doesn't always work or doesn't always produce completely optimal results, but which goes some way towards solving a particularly difficult problem for which no optimal or perfect solution is available.

Inference engine

A rule-based system requires some kind of program to manipulate the rules - for example to decide which ones are ready to fire (i.e., which ones have conditions that match the contents of working memory). The program that does this is called an inference engine, because in many rule-based systems, the task of the system is to infer something, e.g., a diagnosis, from the data using the rules. See also match-resolve-act cycle.

Knowledge base

Collection of the data and rules that suitably represent the problem domain.

Match-Resolve-Act cycle

The match-resolve-act cycle is the algorithm performed by a forward-chaining inference engine. It can be expressed as follows:

loop

match all condition parts of condition-action rules against working memory and collect all the rules that match;

if more than one match, resolve which to use;

perform the action for the chosen rule until action is STOP or no conditions match.

Step 2 is called "conflict resolution". There are a number of conflict resolution strategies.

RETE

Algorithm used to optimize forward chaining inference engines by optimizing time involved in recomputing a conflict set once a rule is fired.

Rule-based system

A rule-based system is one based on condition-action rules.

Search

Search is a prevalent metaphor in artificial intelligence. Many types of problems that do not immediately present themselves as requiring search can be transformed into search problems. An example is problem solving, which can be viewed in many cases as search a state space, using operators to move from one state to the next.

Particular kinds of search are breadth-first search, depth-first search, and best-first search.

Working memory

The working memory of a rule-based system is a store of information used by the system to decide which of the condition-action rules is able to be fired. The contents of the working memory when the system was started up would normally include the input data - e.g., the patient's symptoms and signs in the case of a medical diagnosis system. Subsequently, the working memory might be used to store intermediate conclusions and any other information inferred by the system from the data (using the condition-action rules).

5.10. WG Appendix B: Online References

See References section.

6. Working Group on Goal-oriented Action Planning

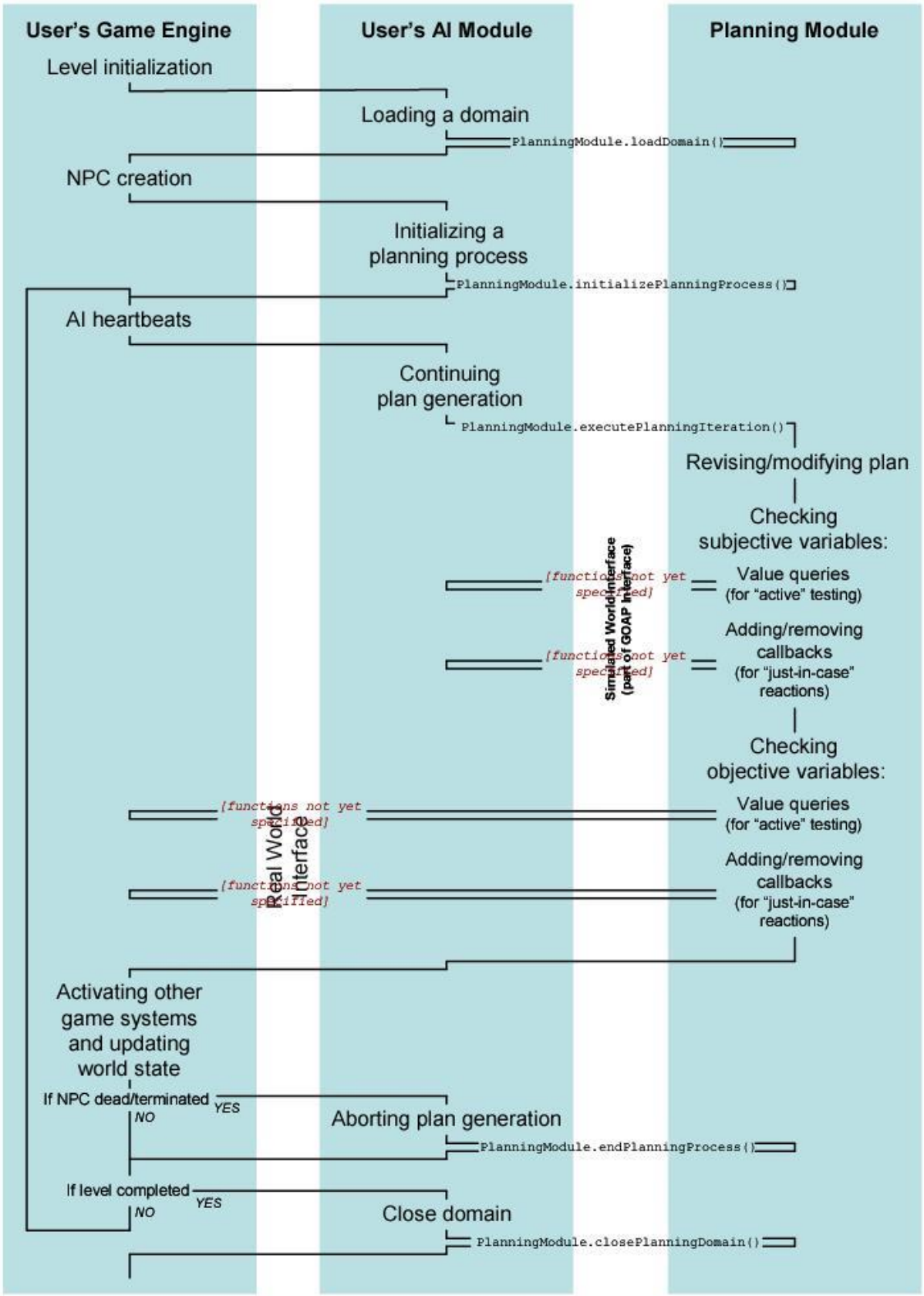
The goal of this working group is to create an interface standard specification for real-time *Goal-oriented Action Planning* (also called *AI Planning* or *Action Planning*), such that a computer game can easily call modules providing planning functionality. A planning system takes as input the current state of the world, goals to be satisfied and possible actions to be executed, and generates a plan, which is a valid sequence of actions that, when executed in the current state of the world, satisfies the given goals. We will not go into further details here, however, as the main purpose of this document is an update about our progress.

Our group has made considerable progress within the last year (over 200 messages were posted to our forum), and we are quite close to an actual interface specification. Many discussions are of course still in progress, and we will only briefly touch such matters in the following.

6.1. General Approach

A planning domain is initially read in. After this has happened, single planning problems can be cast for solution. The solution of a planning problem is an iterative process in which small amounts of computation time are granted for a revision/improvement of the current solution structures (until a solution is found or the process is aborted). The piecewise computation is necessary because the solution of some planning problems requires substantial time. This time cannot be granted in one step because it would hurt other game systems' real-time performance. The stepwise computation, however, requires that external changes/updates of the world that happen in between single planning iterations can be incorporated during a planning process.

For the use of a planning module, i.e., our interface, the following picture provides a preliminary overview/example of the control flow and functions. The specifications of the functions can be found in Section [Interface Functions]. Note that anything related to the use of plans is not yet included.



The domain definition and other information specific to the concrete planning problem will be represented by XML-like data structures (see comments on OWL in Section [Miscellaneous]). In contrast, specification formats like PDDL use a

LISP-like format. We expect, however, that people that use our interfaces will hardly be familiar with LISP.

6.2. Glossary

The following definitions show some glossary updates, which may help you to understand the descriptions. However, we recommend to that you skip it for now and only check it if you actually want to look up a term.

Action
An action is an atomic behavior that contributes to the satisfaction of a goal or multiple goals. Actions have a variable number of effects and preconditions. Effects define how executing the action will change the state of the world. Preconditions define aspects of the state of the world that must be true prior to executing this action.

Effect
Propositions that will be changed when an action is executed.

Goal
A sub-state of a game environment (attached to a character or the world) that a plan is intended to achieve and that when successfully reached terminates the behavior initiated to achieve it.

HTN
Hierarchical Task Network. A form of planning that decomposes high-level tasks into simpler ones [HTN].

PDDL
Planning Domain Definition Language. A standardized action-centered language, inspired by STRIPS and syntactically similar to LISP. Designed as a common way of representing planning problems [fox, pddl].

Plan
A plan is a valid sequence of actions that, when executed in the current state of the world, satisfies some goal or multiple goals. A sequence of actions is valid if each action's preconditions are met at the time of execution. The planner attempts to find an optimal `plan`, according some cost metric per action. The planning process cannot manipulate the actual state of the world. Instead the planner operates on a copy of the world state representation that can be modified as the planner evaluates the validity of all possible sequences of actions.

Planning Domain
A planning domain consists of a set of actions. Each action may be executed only if meets its preconditions, and has some particular set of effects on the state of the world.

Planning Iteration
A partial execution of computing a (partial) plan. A planning module might need multiple planning iterations to compute a (partial) plan. The duration or stopping criteria of a single planning iteration can be configured by the user of a planning module. This configuration is not part of our interface but can be done in a module-specific way.

Planning Module
The part of the code that has to compute (partial) plans, i.e., which provides the planning functionality.

Planning Process

A specific planning problem ("threat") in its current solving state, which is worked on.

Planning Problem
A planning problem consists of specific goals and instantiations of plan variables for a planning domain. A planning process can solve such a problem, i.e., produce a valid plan that satisfies the given goals.

Example: The agent has cooking skills and a kitchen (the planning domain). He wants to cook dinner (the goal) and thinks that there is milk in the fridge (an instantiation of a plan variable).

Precondition
Preconditions define aspects of the state of the world which must be true prior to executing the action. For example, a door must be unlocked before it can be opened.

Real World Interface
The actual state of the world in which an agent belongs. This contrasts with the agents view of the world as defined by simulated world interface

Simulated World Interface
An agents view, or interpretation, of the world in which it belongs. This may not be the actual state of the world, as defined by real-world interface. For example, an agent would believe there to be Milk in the fridge if it was unaware that another agent drank it.

6.3. Interface Functions

In this section, **preliminary** functions for basic interaction with a planning module are specified. Note that the function definitions are still very much under revision, and that you should not worry about pointers vs. real objects etc. Many functions are also still missing, e.g., to inform the planning module about value updates, or functions for checking validity of a plan. We show this **only to give you an approximate idea of our direction**.

Initialization

`PlanningModule.loadDomain()`

This function presents a planning domain to the planning module and requests it to internally store this domain. An identifier for the domain that was loaded is returned.

```
DomainID dID =
  PlanningModule.loadDomain( XMLDomainDefinition dDef );
```

returns: `DomainID dID`

An identifier for the planning domain that was loaded. This identifier can be used in following interactions with the planning module to reference this domain.

`XMLDomainDefinition dDef`

The definition of the planning domain to be loaded.

Example application:

```
XMLDomainDefinition dDef = [structures not yet defined]
DomainID dID = PlanningModule.loadDomain( dDef );
```

`PlanningModule.initializePlanningProcess()`

Generates and initializes a planning process, which is responsible for solving a specific planning problem. An identifier for the planning process that was generated is returned.

```
PlanningProcessID ppID =
  PlanningModule.initializePlanningProcess(
    DomainID dID ,
    [structures not yet defined] vInit ,
    [structures not yet defined] g );
```

returns: `PlanningProcessID ppID`

An identifier for the planning process that was generated. This identifier can be used in following interactions with the planning module to reference this planning process.

`DomainID dID`

An identifier of the planning domain that is to be used for the planning process. This identifier has to be generated by calling `PlanningModule.loadDomain()` before.

`[structures not yet defined] vInit`

This XML structure initializes specific variables of the planning domain.

```
[structures not yet defined] g
```

This XML structure defines the goals that the resulting plan of the planning process should fulfill.

Example application:

```
...
DomainID dID = PlanningModule.loadDomain( dDef );
[missing: generation of vInit]
[missing: generation of g]
PlanningProcessID ppID = initializePlanningProcess( dID , vInit , g );
```

Plan Generation

PlanningModule.executePlanningIteration()

Instructs the planning module to advance the plan generation for one planning iteration. The time constraints for a planning iteration are dependent on the specific planning module (and might be set by middleware-specific functions). It is returned whether a correct plan was found that satisfies the planning process' goals. If this is not the case, it might be possible to generate a plan by repeatedly calling this function.

```
bool planReady =
  PlanningModule.executePlanningIteration( PlanningProcessID ppID );
```

returns: `bool planReady`

If a plan was found that satisfies the planning process' goals, true is returned, and false if a plan could not be generated in this planning iteration.

`PlanningProcessID ppID`

An identifier of the planning process that is to be advanced. This identifier has to be generated by calling `PlanningModule.initializePlanningProcess()` before.

Example application:

```
...
PlanningProcessID ppID = initializePlanningProcess( dID , vInit , g );
...
bool planReady = false;
while( myGameModule.stillEnoughTimeForNPCComputations() && !planReady )
{
  planReady = executePlanningIteration( ppID );
}
```

Result Retrieval

`PlanningModule.retrievePlan()`

Retrieves a plan from the planning module. This plan is only available if the planning module has computed a correct plan (see function `PlanningModule.executePlanningIteration()`).

Note that there is no guarantee that a correct plan exists even if function `PlanningModule.executePlanningIteration()` returned true before because there might have been updates on the world state in the meantime.

```
XMLPlanStructure plan =
  PlanningModule.retrievePlan( PlanningProcessID ppID );
```

returns: *XMLPlanStructure plan*

> Describes the plan that has been computed. The function returns `null` if the planning process did not result in a correct plan yet.

PlanningProcessID ppID

> An identifier of the planning process that has produced the plan. This identifier has to be generated by calling `PlanningModule.initializePlanningProcess()` before.

Example application:

```
...
PlanningProcessID ppID = initializePlanningProcess( dID , vInit , g );
...
bool planReady = false;
while( myGameModule.stillEnoughTimeForNPCComputations() && !planReady )
{
  planReady = executePlanningIteration( ppID );
}

XMLPlanStructure plan = null;
if ( planReady )
{
  plan = PlanningModule.retrievePlan( ppID );
}
```

Cleanup

`PlanningModule.endPlanningProcess()`

Notifies the planning module that a planning process is terminated and all related resources can be released.

```
void PlanningModule.endPlanningProcess( PlanningProcessID ppID );
```

PlanningProcessID ppID

An identifier of the planning process that is being terminated. This identifier will not be valid anymore after the function call.

Example application:

```
...
PlanningProcessID ppID = initializePlanningProcess( dID , vInit , g );
...
PlanningModule.endPlanningProcess( ppID );
```

PlanningModule.closePlanningDomain()

Notifies the planning module that a planning domain is not needed anymore and all related resources can be released. Note that all planning processes of this domain have to be ended before (see PlanningModule.endPlanningProcess()).

```
void PlanningModule.closePlanningDomain( DomainID dID );
```

DomainID dID

An identifier of the planning domain that is to be closed. This identifier will not be valid anymore after the function call.

Example application:

```
...
DomainID dID = PlanningModule.loadDomain( dDef );
...
PlanningModule.closePlanningDomain( dID );
```

6.4. Miscellaneous

- We consider HTN/hierarchical planning functionality as very useful, but will not include it for the first interface version. However, we already keep it in our minds for a future version and pay attention to not already block necessary extensions/variations.
- An alternative to specifying structures in XML is OWL/OWL-Lite/OWL-S, which is currently being discussed.
- Definitions and basic ingredients of what constitutes a planning domain and a planning problem are very much under discussion as well. For example, more complex data types than simple TRUE/FALSE for state variables should be possible, like numbers and objects/links. We also need the power of temporal

quantifiers for states. For now, temporal intervals seem to be the best choice for this.
- We have created and will further detail two examples, situated in the "Sims" domain, which we will mainly use for examples and evaluations: In one example, a Sim has to satisfy his needs regarding hunger, energy, hygiene and going to work, and in the other one, he has to generate a long-term plan to organize his life, like changing his career or living situation.
- The GDC roundtable discussion this year pointed us to other very important issues, which we certainly have to pay more attention to the potential integration with scripting, the need to have a very first action determined as quickly as possible, and graded quality results for plan optimization.

6.5. Group Members

Current members of the working group on goal-oriented action planning:

- *Group coordinator:* Jeff Orkin - Monolith Productions
  (from September 2004 until now, Alexander Nareyek temporarily took over the coordinator role for Jeff Orkin)
- Penny Baillie-de Byl - University of Southern Queensland
- Daniel Borrajo - Universidad Carlos III de Madrid
- John Funge - iKuni Inc.
- Massimiliano Garagnani - The Open University
- Phil Goetz - EOS Logic, LLC
- John J. Kelly III - Model Software
- Héctor Muñoz-Avila - Lehigh University
- Jeff Orkin - Monolith Productions
- John Reynolds - Ubisoft
- Brian Schwab - Sony Computer Entertainment America
- R. Michael Young - North Carolina State University

Further group consultant(s): Richard Evans (Maxis / Electronic Arts)

7. Other Groups (Steering and Finite State Machines)

Unluckily, these groups were not able to finalize a report on their activities by the deadline of this report. The activity of these groups, however, has been very low during the last year.

7.1. Group Members - Working Group on Finite State Machines

- *Group Coordinator:* Nick Porcino - LucasArts Entertainment
- Sam Calis - Universal Interactive
- Scott Davis - Black Cactus Games
- Fred Dorosh - BioGraphic Technologies
- Daniel Fu - Stottler Henke Associates
- Mark Gagner - WMS Gaming
- Ben Geisler - Raven Software / Activision
- Sunbir Gill - Vicarious Visions
- Athomas Goldberg - Sun Microsystems
- Dave Kerr - Naturally Intelligent
- Linwood H. Taylor - University of Pittsburgh
- Alex Whittaker - Beautiful Game Studios

7.2. Group Members - Working Group on Steering

- *(No group coordinator at the moment)*
- Mat Buckland - ai-junkie.com
- Marcin Chady - Warthog
- Philippe Codognet - University of Paris 6
- Mike Ducker - Lionhead Studios
- Leon C. Glover - Entropy Unlimited
- Daniel Kudenko - University of York, UK
- Dave C. Pottinger - Ensemble Studios
- Craig Reynolds - Sony Computer Entertainment America
- Adam Russell - Pariveda

Further group consultant(s): Thaddaeus Frogley (Rockstar Vienna)

8. Support Team

The AI Interface Standards Committee (AIISC) support team consists of people from all around the world that are enthused about game AI and whose main role is, as the name of the group implies, to provide some level of support for the expert groups. Börje Karlsson is the current group coordinator and is the main communication link between the committee chairperson Alexander Nareyek and the support team. Additionally, it is the coordinator's task to try to distribute the workload onto all team members.

Our forum has some postings but most of the work is organized by e-mails sent back and forth between team members or the AIISC working group coordinators. Being far outnumbered by the other committee members - the experts - and all of us studying actively, it is sometimes hard to deliver immediate support when asked for. Nonetheless, we are striving to offer the best help possible.

Sometimes, the group can get quite quiet for a while and so we must take a more pro-active instance and spur new discussions and efforts. Which can always get everything moving again.

As we always say, we are proud to support the world's best game AI experts and to learn from them at the same time. The AIISC is a team, and we are trying to help make a difference with our support - and to have fun besides, too. Overall collaborating with the AIISC working group coordinators and experiencing their passion is really great!

If you're thinking on joining, please contact Alex (the committee chairperson) and let him know, we always need new energetic people. :)

8.1. Tasks

Support group tasks involve:

- summarizing the discussions of the different AIISC working groups,
- creating activity reports of all AIISC members,
- developing support software,
- testing and proposing new tools that can be used to enhance the committee's productivity,
- assisting in the creation of presentation slides and reports,
- helping with technical problems mostly concerning the usage of SourceForge.net and accessing its CVS repositories,
- working on the creation of a glossary to try to uniform terminology usage among the groups,

- and having an open ear to all needs and problems that might occur in the daily committee work.

Every member monitors at least one committee discussion forum to try to react quickly on support demands and to protocol the posted arguments in summaries. In general, we do not engage actively in the discussions, only sometimes when we think that we could provide some expertise as well or some of point of view regarding the issue at hand.

The current approach to writing the summaries is to try to have more than one member monitoring each forum. One will produce the summary and one of the others will be something like a reviewer, changing roles the next time. With the steadily growing number of postings (and concentration of activity in specific groups at a time), more and more group work will surely have to take place, mainly between the supporters assigned to the same AIISC working group.

The support team has also started working on issues of the support activity itself. We are on an ongoing effort to try to develop guidelines for making summaries, approaching group coordinators and quickly integrate and evaluate our work. This way we expect to raise our productivity and especially that of new members - also in a more organized way :). Along with that effort, we are also trying to find new ways in which we can help the experts.

8.2. Group Members

Current members of the support team:

- Alex J. Champandard - University of Edinburgh
- Bjoern Knafla - University of Kassel
- Börje Karlsson - Pontifícia Universidade Católica do Rio de Janeiro *(current group coordinator)*
- Cengiz Gunay - Emory University
- Eric Martel - Microids
- Hugo da Silva Sardinha Pinto - Universidade Federal do Rio Grande do Sul
- Jayaraman Ranjith - International Institute of Information Technology
- José Lopes - University of Exeter
- Samir Pipalia - City University, London
- Stephen D. Byrne - Hiram College

References

[1] C. L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 19 (1), 1982.

[2] J. C. Giarratano and G. D. Riley. Expert Systems: Principles and Programming, Second Edition, PWS Publishing, 1993.

[3] JCP. JSR 94: Java Rule Engine API, 2004. URL: https://www.jcp.org/en/jsr/detail?id=94

[4] eXpertise2Go. Introduction to Expert Systems, 2001. URL: https://www.expertise2go.com/webesie/tutorials/ESIntro/

[5] Alison Cawsey. Forward Chaining Systems. In Databases and Artificial Intelligence 3 - Artificial Intelligence Segment, 1994. URL: https://www.cee.hw.ac.uk/~alison/ai3notes/subsection2_4_4_1.html

[6] Alexander Nareyek, Bjoern Knafla, Daniel Fu, Derek Long, Christopher Reed, Abdennour El Rhalibi, and Noel S. Stephens. The 2003 Report of the IGDA's Artificial Intelligence Interface Standards Committee. International Game Developers Association (IGDA), Tech Report, June 2003.

[7] Alexander Nareyek, Börje F. Karlsson, Ian Wilson, Marcin Chady, Syrus Mesdaghi, Ramon Axelrod, Nick Porcino, Nathan Combs, Abdennour El Rhalibi, Baylor Wetzel, and Jeff Orkin. The 2004 Report of the IGDA's Artificial Intelligence Interface Standards Committee, International Game Developers Association (IGDA), Technical Report, June 2004.

[8] Dana S. Nau. Hierarchical Task Network Planning. In: Automated Planning: Theory and Practice. University of Maryland, 2004. URL: https://www.cs.umd.edu/~nau/cmsc722/notes/chapter11.pdf

[9] Maria Fox. PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research (JAIR), 2003. URL: https://www-2.cs.cmu.edu/afs/cs/project/jair/pub/volume20/fox03a-html/node2.html

[10] S. Edelkamp and J. Hoffmann. PDDL 2.2: The Language for the Classical Part of the 4th IPC. URL: https://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/DOCS/pddl2.2.ps.gz

[11] Khronos Group. COLLADA 1.3.1 Specification Collada. June, 2005. URL: https://www.khronos.org/collada/