Proyecto GKAPPA

Comentarios basados en la presentación del proyecto GKAPPA realizadas por Adrían Biga, en las 3eras. Jornadas de Ciencias de la Computación.

1-Motivacion a realizar el proyecto

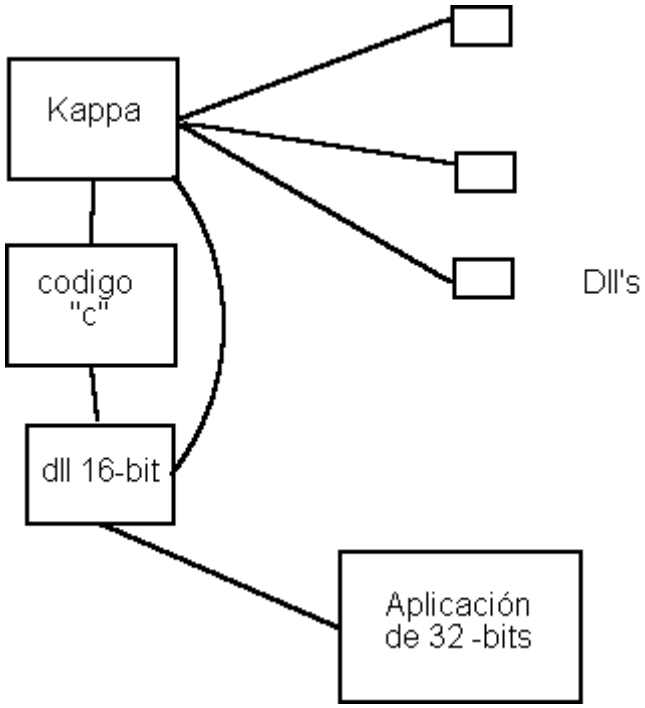
1-1 Limitaciones y virtudes de Kappa

Una pregunta que se nos puede plantear es porque no utilizamos otras herramientas libres y conocidas con tecnología de 32-bits (ejemplos: JESS, CLIPS). Para poder contestar la misma debemos entender los comienzos de un proyecto llamado Endodiag + y su propósito.
Antes de comenzar con el proyecto Endodiag +, existen 2 versiones anteriores a este denominados Endodiag y Endodiag II
Endodiag es un sistema experto desarrollado en conjunto por la cátedra de endodoncia de la Fac.de Odontología y el Dpto. de Sistemas e Informática de la Fac. de Ingeniería y Cs.Exactas. El desarrollo de este primer sistema tuvo como objetivo apoyar el aprendizaje diagnóstico en el área de Endodoncia. Los alumnos interactuaban con él presentando un caso clínico que analizaban con el apoyo del sistema para arribar al diagnóstico adecuado.
Endodiag se utilizó con alumnos de la carrera de especialización en Endodoncia, y se vio entonces la necesidad de enriquecer el desarrollo con la incorporación de casos clínicos que el mismo sistema presentara a los alumnos, adaptando su comportamiento a las características y conocimientos de los estudiantes. Se encara entonces el desarrollo de un segundo sistema, llamado endodiag2, que incorpora una base de casos clínicos representados por situaciones reales de la practica profesional, mas la construcción y utilización de perfiles de usuario, que almacenan la evolución del alumno y seleccionan el caso mas adecuado para colaborar en la formación del mismo. (En este sistema trabajó Andrea Torres con la realización de su tesina).
Al evaluar este segundo sistema, se planteó agregar funcionalidad relacionada a tareas de tutoría on line, posible a partir de la integración a la WEB. A partir de la complejidad agregada y de las distintas funciones diferenciadas en el sistema se pensó entonces en migrar a una arquitectura MAS. Sin embargo, los sistemas ya existentes contienen una ingente cantidad de conocimiento y una funcionalidad adecuada a los objetivos propuestos al momento de encarar su desarrollo, y por lo tanto es importante poder reutilizar el trabajo realizado a través de varios años.
Por lo tanto el desafío es escalar el sistema existente a una arquitectura multiagente, reutilizando el motor de inferencia y la base de conocimientos ya implementados.
Kappa es una herramienta desarrollada hace ya muchos años con una arquitectura y que ha quedado desactualizada como resultado de que la empresa propietaria discontinuó su comercialización y actualización a fines

de los 90. Sin embargo, liberó el código para su libre uso y esta situación abre la posibilidad de modificarlo con el fin de poder reutilizar los desarrollos existentes.

1-2 Motivación técnica

En este momento, KAPPA-PC es un software propietario de tecnología de 16-bits de más de 13 años.
En el 2002 este producto ha sido discontinuado y dejado bajo licencia GPL, por lo cual se tiene un software libre pero solamente compuesto por su código fuente de 16-bits debido a que las herramientas utilizadas para generar las librerías y los ejecutables son también propietarias.
En la actualidad para poder realizar una aplicación de mas alto nivel, que por ejemplo utilice el motor de inferencia se debe compilar el código c



"Figura 1: Kappa Hoy"

generado por Kappa y de esta manera embeber la funcionalidad en una dll de 16-bits que a su vez utiliza las funciones encapsuladas en las dll que componen, KAPPA-PC.Por lo tanto si deseamos desarrollar una aplicación de 32-bits (tecnología actual) que requiera el uso de Kappa se deben desarrollar parches utilizando técnicas de thunking abstracto, genérico o directo para poder tener acceso a la funcionalidad requerida(observar figura 1).
Lo que en definitiva representa una resolución a un solo problema en general utilizando técnicas avanzadas de programación que no siempre funcionan correctamente (dependen del entorno de desarrollo, SO, etc.)
Por lo que se propone una solución directa y definitiva a esta problemática, lo que permitirá el desarrollo de aplicaciones basadas en Kappa con tecnología de 32-bits y de una manera casi automática.

Por lo cual se penso en la alternativa de utilizar el código abierto y empezar a migrar y adaptar este código, generando las librerías que componen el sistema pero a partir de ahora en una tecnología de 32-bits. (ver figura 2)

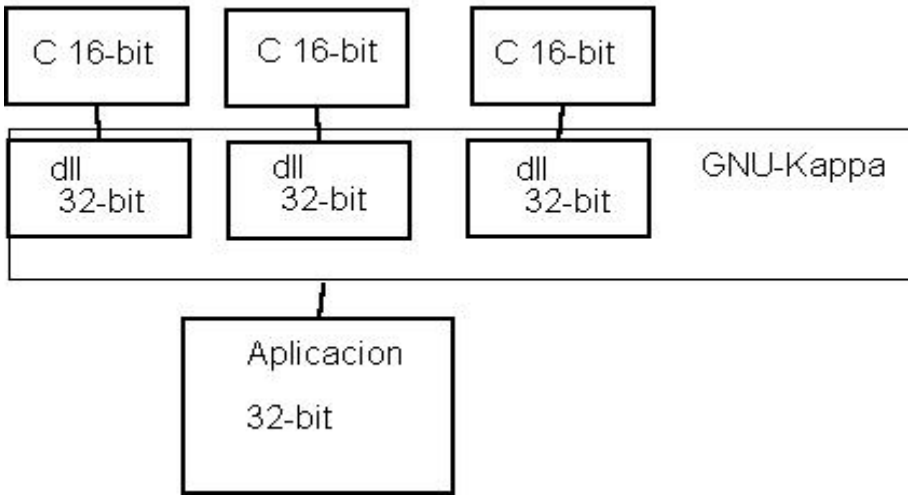
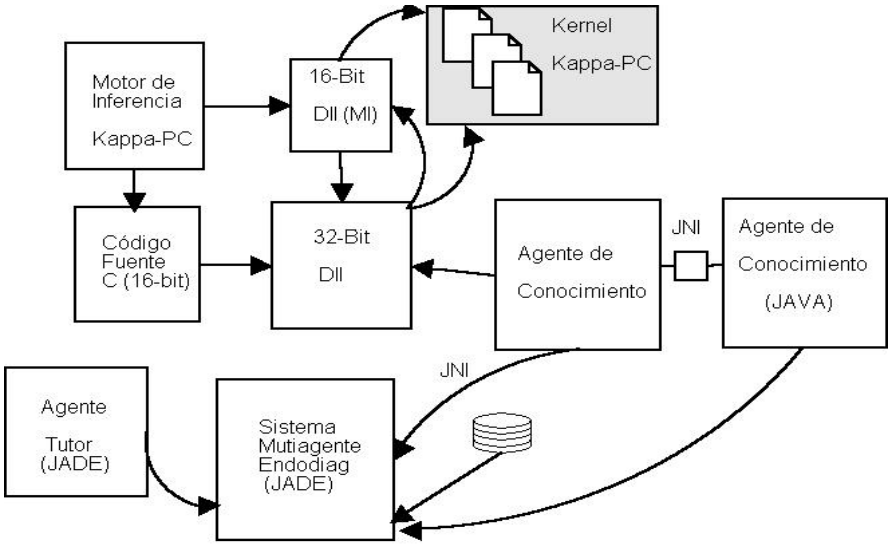


Figura 2: GNU-Kappa

1-3-Proyecto Endodiag +

El proyecto Endodiag + al utilizar el conocimiento y el motor de inferencia del producto Kappa para poder integrarlo con tecnología de 32 bits requerida al seleccionar como sistema plataforma multiagente al sistema JADE, debía utilizar técnicas de programación denominadas thunking para poder realizar llamadas de aplicaciones de 32 –bits a aplicaciones o dlls de 16 bits que integran la funcionalidad de los programas generados por el compilador de Kappa.

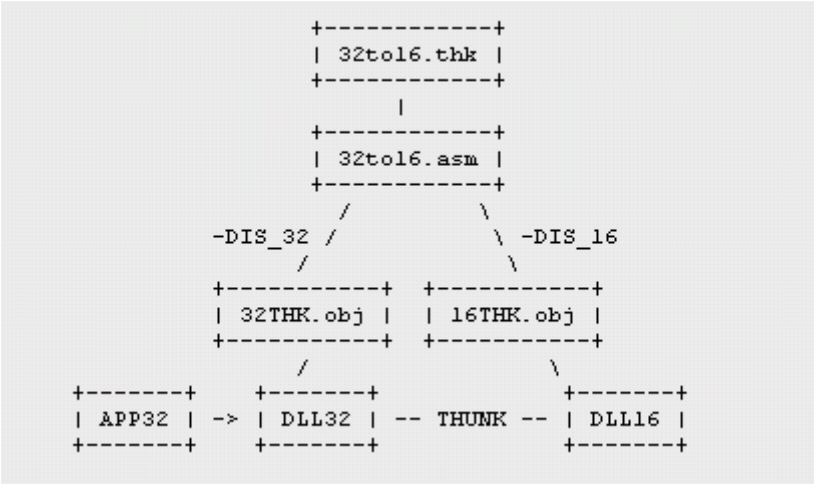


"Posible Solución al problema "

2-Un poco de la técnica de “Thunking”

Veamos la problemática que se presenta al usar esta técnica.
Desgraciadamente arrastramos un código de tecnología de 16 bits. Una de las soluciones para llamar funciones desde código de 32 bits a funciones de 16 bits (o viceversa) se denomina “Thunking” (es un parche, en definitiva).
Existen tres técnicas distintas de la misma:
-FLAT: Win95-Win98-WinMe
-GENERIC: WIN2000-NT, XP
DIRECT: Win95-Win98-WinMe -. Por lo tanto no son compatibles!!!
Para poder utilizar estas técnicas (por lo menos las dos primeras) requerimos de un compilador llamado “THUNK” no libre, conocimientos de assembler y mucho tiempo para prueba y error(es un código difícil de debuggear).
Al terminar el proceso se finaliza con al menos dos dlls una de 16 bits y otra de 32 bits.
Una técnica más sencilla de utilizar, (a veces, se comienza a complicar en la utilización de parámetros de tipo puntero) es la técnica que se denomina “thunk directo”. La cual no necesita usar el compilador thunk y se basa en llamar a la función que posee la kernel32.dll “QT_THUNK “(por eso no hay compatibilidad, no todas las distribuciones de WINDOWS poseen esta función).
Esta técnica no es tan simple cuando se desean pasar punteros de aplicaciones de 32 a aplicaciones de 16 bits; por lo tanto a veces es conveniente volver a las técnicas FLAT o GENERIC dependiendo del caso de uso.
Además de la problemática que se nos presenta en las primeras dos técnicas, de tener que usar el compilador thunk, la dificultad del debuggeo, la poca portabilidad de los parches generados, etcétera.
Un thunk consiste en 2 dll una de 32 y otra de 16, en las llamadas de 32-16 la dll de 32 llama a una función exportada en la dll de 16 bits. Por lo tanto necesitamos un compilador de 16 bits, uno de 32 bits, el compilador thunk y el MASM por ej. , este ultimo para ensamblar los.asm generados por el compilador thunk.

Veamos los pasos a seguir para aplicar esta técnica:



1-Crear un thunk script.thk
2-Compilarlo con el "THUNK", generando.asm
3-Ensamblar para generar los .OBJ
4-Los módulos obj se deben linkear a las respectivas dlls.

Bueno si esto, no es ya demasiado, esta técnica hereda las limitaciones propias de la tecnología de 16 bits WINDOWS. (Por ej. Segmentación, no se considera multitarea, etc.)

3-Migracion del código de Kappa - Proyecto Gkappa

Al observar las dificultades y limitaciones que nos genera la utilización de la técnica anterior y al poder contar con el código fuente de la versión de 16 bits decidimos realizar la migración del software Kappa-Pc al software Gkappa. Lo que impone el nacimiento de este proyecto.
Los beneficios de este nuevo proyecto:
-se obtendría en una primera fase el motor de inferencia totalmente en una tecnología de 32 bits y con licencia de código abierto (gpl).
Lo cual facilitaría el desarrollo casi-automatico de sistemas de alto nivel que usen el motor (ahora es muy artesanal)
-se brindaría este código a toda la comunidad GNU vía la utilización de la web y seria fomentada por la cátedra, lo que proporcionaría el acercamiento de esta "comunidad".
-se brindaría un software totalmente desarrollado por la cátedra y mantenido vía web por la misma (único en nuestra carrera)
-se estaría cumpliendo con la migración de un software que ha sido discontinuado de producción comercial.
-se obtendría un software modular de código abierto, para que cualquier alumno lo adapte a sus necesidades.
Veamos un poco, el desafío al que nos encontramos actualmente. Para lo cual veamos algunos cuestiones que deben tenerse en cuenta a la hora de migrar la arquitectura.

Diferencias generales de las dos tecnologías:

Los punteros son de 32 bits se deja de utilizar en el código declaraciones como _near o _far.
Los handles de ventana por ej. son también de 32 de bits por lo tanto no se pueden intercambiar con otros tipos como se realizaba en 16- bits, utilizar indistintamente WORD O HWND, ahora son tipos de longitud diferente. Ahora se presentan en formato de 16 y 32 respectivamente.

Problemas Generales:

-declaraciones de procedimientos de ventana

-declaraciones de punteros (near y far)

-tipos de datos

-mensajes

-llamadas a las api del sistema (las api están desactualizadas, no son las mismas en los 2 sistemas sol:
thunking ¿? no obviamente)

-función winmain (cambia la declaración y/o definición ¿?)

El incremento de los punteros a un tamaño de 32 bits genera uno de los más típicos problemas a la hora de realizar una migración. Este problema puede afectar la parametrización de mensajes por ej. si se asumen distintos tamaños.
La migración de tecnologías requiere de un análisis completo del código fuente del software (sol: herramientas que automaticen la búsqueda de problemas), por lo tanto es necesario definir las técnicas a seguir en este camino, por ejemplo podríamos adoptar un camino de tipo top-down. Llamamos top-down porque intentamos encontrar los errores a partir de la compilación del código y luego proceder a aplicar la técnica de migración apropiado, una metodología de tipo buttom-up seria, localización en el código, planteo de la solución y luego compilación por ejemplo. La primer metodología sería básicamente obtener los problemas utilizando un compilador de 32-bits y comenzar a tratarlos según ciertas reglas.

Ejemplo sencillo en el código de Kappa-PC:

(Presente en el código de generación de la dll ->kaprule.dll)

```
short NEAR RulePriority (ITEMID idRule)
{
  LPRULE lpRule;
  short  sRet;
```

```
    if ((lpRule = (LPRULE) KppGetItem (RULE, idRule)) == NULL)
      return 0;
    sRet = PRIORITY (lpRule);
    KppReleaseItem (RULE, idRule);
    return sRet;
  }
```

el nuevo código se vería de esta forma:

```
  short  RulePriority (ITEMID idRule)
  {
    LPRULE lpRule;
    short  sRet;
    if ((lpRule = (LPRULE) KppGetItem (RULE, idRule)) == NULL)
      return 0;
    sRet = PRIORITY (lpRule);
    KppReleaseItem (RULE, idRule);
    return sRet;
  }
```

Near se remueve debido a que en tecnología de 32 bits, far y near son removidos debido a la arquitectura.
Short es de 16 bits tanto en la tecnología de 16 bits como en la de 32,
Si en vez de short tendríamos int si tendríamos una complicación debido a que en 32 es de 32 bits, no así en la de 16 que es de 16 bits. Y NULL?

5-Trabajos futuros y estado del arte.

El estado del arte en este momento:

- Finalización de la metodología a adoptar para realizar la migracion, punto visto anteriormente. (investigación)
- Construcción de herramientas para la detección de problemas.
- Aislar el motor de inferencia, para implementar el primer paso de la migración.
- Generación del sitio y la documentación.

Trabajo Futuro:
-Completar la migración incluyendo la del compilador de KAPPA.
-Portabilidad de Kappa a sistemas Linux.

6-Bibliografia

- 1. Departamento de Sistemas e Informática Facultad de Ciencias Exactas, Ingeniería y Agrimensura – UNR. Proyecto GKAPPA.
- 2.Casali A.,Corti R.,D' Agostino E. ,Biga A.,Siragusa M.,Aciar S.Sistema de apoyo al aprendizaje diagnóstico: de Endodiag a un sistema multiagente.

- 3.Casali A., Corti R., D'Agostino E., Siragusa M. Sistema Basado en Conocimiento de Apoyo al Diagnóstico de la Patología Pulpar y Periapical. Anales 31 JAIIO, Actas de SIS – ISSN 1666-1125, vol 5, pp 192-196. Santa Fe, 2002.
- 4. Casali A., Corti R., D'Agostino E., and Siragusa M.. Herramienta tecnológica como apoyo al diagnóstico endodóntico. Electronic Journal of Endodontics Rosario Ejer. ISSN 1666-6143 Año 3 - Volúmen 1, 2004.

5-MSDN.www.msdn.com

6-"Undocumented Windows 2000 Secrets" - Sven V. Schreiber
7-"Windows 95 System Programming Secrets" - Matt Pietrek

-Contactos:

-Rosa Corti - rcorti@fceia.unr.edu.ar

-Ana Casali - acasali@fceia.unr.edu.ar

-Adrián Biga – abiga@fceia.unr.edu.ar