

Congreso Iberoamericano de Ciencia, Tecnología, Sociedad e Innovación CTS+I

Palacio de Minería del 19 al 23 de Junio de 2006

Los géneros de discurso como un fundamento epistémico del software

SERGIO ELLERBRACKE ROMÁN

MESA 3

Resumen

Se analiza la interpretación de Jean-François Lyotard de los géneros de discurso como un fundamento epistémico del software. Se cuestiona la fundamentación "ortodoxa" del software, basada en las matemáticas.

Dicha fundamentación "ortodoxa" de la computación asume a las matemáticas como al cuerpo de conocimiento "fundante" de la computación. Aquí se parte de la premisa de que el software puede ser interpretado, de manera más fecunda, por medio de constructos lingüísticos.

En el origen mismo de la computación aparece un juego matemático conocido como "la máquina de Turing". La máquina de Turing era capaz de ejecutar cualquier algoritmo matemático, proveniente de cualquier teoría matemática, y fue la base de la arquitectura de las computadoras. Digamos que cualquier proposición matemática puede ser "traducible" a software.

Para el filósofo del lenguaje Jean-François Lyotard, una característica fundamental de un lenguaje es ser traducible. Así, cualquier sentencia del español se puede traducir al inglés. No ocurre lo mismo con los géneros de discurso. Por ejemplo, una proposición jurídica sólo es inteligible dentro de dicho género, y no es traducible al género poético, al económico, al médico, al matemático o al lógico. Cada género de discurso opera de acuerdo con sus propias reglas y referentes. Cada género de discurso se constituye así en una isla de significación, donde se hace posible la comunicación.

Hay un paralelismo importante entre los géneros de discurso y las teorías matemáticas: en ambos casos, una proposición sólo es significativa dentro de su entorno. Por ejemplo, una proposición de lógica matemática no puede ser traducida a una proposición geométrica, o a una proposición algebraica: sólo es válida dentro del discurso de la lógica matemática. Pero cualquier proposición matemática es representable por medio de software, y cualquier proposición lingüística es traducible en un lenguaje. Así, el lenguaje y el software se pueden interpretar como infraestructuras o ambientes capaces de representar proposiciones de diferentes cuerpos de semántica.

Pero todavía más, el software claramente no se restringe a la representación de proposiciones matemáticas: así lo atestigua la gran penetración de los sistemas de información en las empresas, codificando los procesos administrativos, decididamente compuestos por proposiciones lingüísticas no matemáticas. Una revisión preliminar de las proposiciones lingüísticas capaces de ser traducidas a software, incluiría a las proposiciones performativas, prescriptivas, ostensivas, interrogativas y nominativas, para excluir por ejemplo a proposiciones emotivas, lúdicas y persuasivas.

Introducción: Los géneros de discurso y la lingüística computacional

La lingüística computacional es una disciplina científica de las ciencias de la computación. Por el título de este trabajo sería lógico pensar que debería encuadrarse dentro de la lingüística computacional, y usar el bagaje científico desarrollado en dicha disciplina durante décadas.

Sin embargo, este trabajo es exploratorio. Y ¿cómo justificar un trabajo exploratorio cuando en 1998, nada más en el congreso anual de la ACL (*Association for Computational Linguistics*) se presentaron 251 trabajos? Y cuando además la ACL es una asociación de viejo cuño, y sólo en los últimos 25 años ha logrado acumular casi mil setecientos trabajos, en más de once mil páginas.

Y todavía más, este trabajo lleva el término “fundamento epistémico” en el título, cuando la lingüística computacional se desarrolló bajo el cobijo de las teorías de uno de los más grandes teóricos de la lingüística del siglo XX (Chomsky y su gramática generativa).

Por todo lo anterior, es necesario explicar por qué es posible un trabajo epistemológico-exploratorio a esta altura del desarrollo de la lingüística computacional.

Revisemos el origen de la Lingüística computacional. En Junio de 1952, se realizó la *M.I.T. Conference of Mechanical Translation*¹, que reunió a un grupo de lingüistas e investigadores de la computación interesados en explorar los problemas que implicaría la hipotética traducción automática de un lenguaje natural a otro.

Una vez que editaron en un libro los catorce trabajos de la conferencia del M.I.T., comenzó a imprimirse en 1954 el *Journal Mechanical Translation*, que fue el antecesor de la *Association for Machine Translation and Computational Linguistics* (AMTCL), fundada en 1963, y que cambió su nombre a la *Association for Computational Linguistics* (ACL), en 1968.

El punto interesante es que la lingüística computacional surge alrededor de un problema endiabladamente complejo: la traducción automática. Y la rama de la computación que se enfrenta a esos problemas tan ambiciosos es la Inteligencia Artificial (IA).

Pero además, la traducción automática llevó rápidamente a un problema todavía más grande: la comprensión del lenguaje por la computadora, y lo más importante, dicha comprensión del lenguaje debía llevar a una revolución en la lingüística. En 1963, Victor Yngve (el primer presidente de la AMTCL), escribía:

“Imagina lo que sucedería si tuviéramos programas de computadora que entendieran el inglés. Independientemente de sus obvias implicaciones prácticas, las implicaciones para nuestra comprensión del lenguaje serían excitantes. Esta investigación promete llevarnos a nuevas comprensiones acerca de la manera en la cual los lenguajes manejan la información, la forma en la cual la gente entiende el inglés, la naturaleza de los procesos del pensamiento, la naturaleza de nuestras teorías, ideas y prejuicios, y eventualmente una mayor comprensión de nosotros mismos. ... la interacción con la lingüística ya ha producido pequeñas revoluciones en la metodología, puntos de vista, comprensión del lenguaje y estándares de rigor y exactitud.

¹ Yngve, 1982, p. 92.

Parece que antes de que lo logremos, la lingüística será completamente revolucionada"²

Y lo más importante desde la perspectiva epistemológica es la crítica al estado de la cuestión de la lingüística. Según el multicitado Victor Yngve: "*debemos trascender la tradición gramatical de dos mil años como el núcleo del pensamiento lingüístico, y reconstruir la lingüística sobre principios científicos bien conocidos*"³.

Resumiendo, los temas que aborda la lingüística computacional son también problemas de la Inteligencia Artificial enfocados a la comprensión del lenguaje o a la traducción de lenguajes naturales. Los principales problemas:

1. Revisión ortográfica y sintáctica de textos introducidos por los usuarios de un programa.
2. Búsqueda inteligente de información en textos.
3. Clasificación automática de textos.
4. Reconocimiento óptico de caracteres, tanto escritos a mano como incluidos dentro de imágenes.
5. Generación automática de mensajes.
6. Tratamiento de lenguajes naturales no alfabéticos (p. ej. chino).
7. Reconocimiento de voz.
8. Procesamiento de mapas conceptuales en dominios restringidos (p. ej., la medicina, el derecho).
9. Construcción de diccionarios eficientes.
10. Particularidades gramaticales de los lenguajes naturales.
11. Detección de ambigüedades lingüísticas.
12. Extracción de morfemas.
13. Inferencias temporales a partir de textos.
14. Construcción de programas (*parsers*) que analizan la estructura lingüística de un texto.

Entonces, la lingüística computacional pretende interpretar el lenguaje usando la herramienta computacional, y en el proceso, transformar a la lingüística. Aquí, el sujeto de investigación es el lenguaje, y la herramienta es el software.

Por el contrario, este trabajo pretende sondear al software en sus características lingüísticas, asumiendo que los géneros del discurso pueden ser utilizados como fundamento epistémico del software. Aquí, el sujeto de investigación es el software, y la herramienta es la lingüística.

² Yngve, 1982, p. 94

³ Yngve, 1982, p. 94

1. Características lingüísticas del software

Se procederá a analizar lingüísticamente el software, escalando de los elementos más sencillos (nombres), a las líneas de código, las funciones, los programas y el software en su totalidad.

1.1. Nombres de variables y nombres de funciones

1.1.1. Estilo elíptico

Para ilustrar, Wittgenstein⁴ da como ejemplo de una oración elíptica al grito de un albañil ¡Losa! a su ayudante, en vez de la oración ¡tráeme una losa!

Por ejemplo, algunas variables extraídas de un sistema bancario son PagoGastosSinIntereses, CapitalComercialApertura, PagoInteresVencidoFondo, GarantiasGuardaEntrada, PagoGlobalXQuebranto. Del mismo sistema se extrajeron los nombres de funciones Rescatalo(), CalculaPenasYMoras(), Abonale(), PasaloAContenciosa(), PasaloAVencida() y GetTaAmo(). En software no se acostumbra escribir oraciones completas, sino más bien sustraer elementos claves, y armar nombres con dichos elementos. De hecho, es común truncar las palabras (como en GetTaAmo() que podría haberse nombrado como ObtieneTablaAmortizacion()).

El programador puede construir nombres muy diversos. Por ejemplo, la función PasaloAContenciosa() podría haberse nombrado como PasaElCreditoVencidoAJuridico(), o simplemente como Contenciosa(), o como Contenc(), o simplemente Cont(), y en cualquier caso auxiliarse con comentarios. Así la función podría ser

```
// Cont toma el crédito actual, que debe tener más de tres mensualidades vencidas
// para entrar en esta función, a estatus contencioso (que pague o se hacen válidas
las
// garantías).
void Cont()
{
...
}
```

1.1.2. Palabras compuestas

Con el mismo conjunto de variables y nombres de funciones es fácil observar el uso de palabras compuestas (en alemán, *Kompositum Word*). De la misma manera en que en alemán es típico crear palabras compuestas (por ejemplo, *Wirtschaftsingenieurwissenschaften*, que significa Ingeniería Industrial, y se compone de Wirt (economía), Ingenieurs (ingeniería), wissen (saber), y schaft, que transforma un verbo en sustantivo), en el software básicamente se trabaja por medio de palabras compuestas. Por supuesto, un programador puede crear nombres de variables o funciones de longitud minúscula, pero se considera una mala práctica profesional.

⁴ Wittgenstein, 1958, p.33

1.1.3. Ausencia de pronombres personales

Wittgenstein comenta el hecho que el discurso físico se caracterice por no emplear pronombres personales⁵. En el caso de la física, se da por la tradición de no redactar textos científicos en primera ni en segunda persona, pero en el caso del software no se utilizan pronombres personales porque las líneas de código no forman parte de una conversación entre personas, sino que son codificadas por un programador y ejecutadas dentro de una computadora, sin intervención humana. El código del software no es un dialógico.

1.1.4. Ausencia de nombres propios

Casi siempre, los nombres de variables o los nombres de funciones utilizan sólo nombres comunes. Sería bastante raro incluir en el nombre de una variable el nombre de una ciudad, el nombre de una persona concreta, o el nombre científico de una planta. Eso tal vez sea porque una variable puede contener una gama de valores, y es muy difícil que esté referenciada a una persona o ciudad concreta. Por ejemplo, la variable NombreCliente probablemente contendrá (en distintos momentos) los nombres de todos los clientes de una empresa, de manera que carece de sentido el uso de nombres propios. Una excepción podría ser el nombre de la función Poisson(), que calculara la distribución estadística que lleva el nombre de este matemático francés. Pero aquí se usa este nombre propio para realizar un cálculo matemático, no en referencia a la persona. La función podría denominarse DistribucionPoisson(), pero muy difícilmente se denominaría DenisPoisson().

1.1.5. Sustantivos y verbos

Es muy común que los nombres de variables incluyan sustantivos, mientras que los nombres de funciones incluyan un verbo y sustantivos, en ambos casos eslabonados por medio de palabras compuestas. Algunos ejemplos de nombres de funciones tomados del mismo sistema bancario son ValidaFechas(), ChecaFebrero(), AjustaCapitalAde(), CalculaInteres(), ActualizaFechaVencB(), RevisaLiquidacion(), AbonaCapitalVigente() y AplicaMovimiento().

Ahora bien, en el software se tratan de manera diferenciada distintos tipos de verbos. En primer lugar, en el software no se utilizan ni los verbos psicológicos ni los verbos exclusivamente pronominales. Los verbos psicológicos son aquellos que designan acciones, estados o procesos anímicos, mentales o cognitivos (amar, gustar, preocupar, desear, creer, sentir)^{6,7}, mientras que los verbos exclusivamente pronominales expresan procesos que suceden en el sujeto, y se conjugan obligatoriamente con un pronombre (arrepentirse, quejarse, jactarse, dignarse)⁸. Se plantea que la ausencia de ambos tipos de verbos es debido tanto a la ausencia de pronombres personales, como al hecho de que el software no es dialógico.

La inmensa mayoría de los verbos son predicativos (o verbos plenos), que pueden ser transitivos o intransitivos. Los verbos predicativos normalmente se incorporan en el nombre de funciones. Los verbos predicativos encierran la idea de un predicado y siempre expresan el estado o la acción del sujeto al que se refieren⁹.

⁵ Wittgenstein, 1958, p. 299

⁶ Wittgenstein, 1958, p. 505

⁷ educ.ar, 2006.

⁸ Fernández López, 2006.

⁹ Fernández-López, 2006.

1.2. Líneas de código

El equivalente lingüístico de una línea de código es la proposición u oración. Una revisión preliminar de las preposiciones lingüísticas capaces de ser traducidas a software, incluiría a las proposiciones performativas, prescriptivas, ostensivas, interrogativas y nominativas, para excluir por ejemplo a proposiciones emotivas, lúdicas y persuasivas.

1.2.1. *Proposiciones performativas*

Una proposición performativa “hace algo al mismo tiempo que lo enuncia, no sólo describe algo, sino que constituye algo”¹⁰. Las proposiciones lingüísticas no se introducen en un software para entender un fenómeno, para describirlo, sino para hacer algo: automatizar un proceso, hacer una tarea en menos tiempo, aumentar la eficiencia. La codificación del software es una actividad costosa, por lo que normalmente sólo se codifica cuando el análisis costo/beneficio es favorable. Entonces, el software normalmente no se hace “porque sí”, sino con un objetivo, normalmente económico. Por eso, lo natural es que el software “haga algo”.

1.2.2. *Proposiciones ostensivas*

La definición ostensiva explica el uso de una palabra¹¹. La proposición ostensiva, es decir, la mostración del caso, es al mismo tiempo una alusión a lo que no es el caso¹². En el software, los registros indican lo que es el caso señalando los componentes de ese caso, y sólo esos componentes, porque lo que no forma parte del registro, no es ese caso. Por ejemplo, la declaración (en lenguaje de programación C),

```
struct {  
    int      Concepto;  
    long     Credito;  
    long     FechaDesde;  
    long     FechaHasta;  
    double   Tasa;  
    double   Importe;  
} Movimiento;
```

Indica que *Movimiento* consta de un concepto (por ejemplo, los intereses normales), para un cierto crédito, abarcando un cálculo (que va desde la *FechaDesde* hasta la *FechaHasta*, calculado usando una tasa determinada, y por una cierta cantidad de dinero. Lo que es *Movimiento* está señalado ostensivamente, y nada más de lo señalado forma parte de *Movimiento*.

1.2.3. *Proposiciones interrogativas*

En el software no se usa un signo para la interrogación, ni hay un cambio en la entonación para preguntar por algo. ¿Cómo pregunta el software? Pidiendo cosas. Se las pide a otros software o se las pide a personas. En el primer caso un software pide un servicio o algunos datos a otro software, y el otro software proporciona lo que se le ha pedido. En el segundo caso, pide que se llenen ciertos “controles” (por ejemplo, los *edit box*) y no permite avanzar hasta que no se ha proporcionado la

¹⁰ Miguelez, 2001.

¹¹ Wittgenstein, 1958, p. 47.

¹² Lyotard, 1999, p. 62.

información solicitada. Por ejemplo, en una "Terminal Punto de Venta" y al final de una compra, el cajero/vendedor nos pregunta ¿Su pago es en efectivo o en tarjeta de crédito? Si le entregamos efectivo, introduce la cantidad en la computadora, se abre el cajón del dinero y se indica el cambio en la pantalla. Si le damos la tarjeta de crédito, la desliza en un lector adecuado, el software recibe el número de tarjeta y pide la autorización al banco. En cualquier caso, se ha respondido a la pregunta hecha por el software.

1.2.4. *Proposiciones nominativas*

Una proposición nominativa es aquella que nombra un referente. Por ejemplo, el referente *Movimiento*, fue nombrado en el segmento de código mostrado arriba. A veces, se conserva la proposición nominativa que dio origen al nombre de una variable, guardándola como comentario. Por ejemplo,

```
long FechaDisposicion;    // Fecha en que el cliente dispuso (obtuvo) el importe del crédito
```

1.2.5. *Proposiciones prescriptivas y normativas*

Son las proposiciones de la ética y el derecho. De lo que se debe hacer y lo que es ordenado. Las proposiciones prescriptivas proceden del criterio justo/injusto¹³, es el reino de la ética. Las proposiciones normativas son normas o leyes que ordenan lo que se tiene que hacer, es el reino del derecho.

En cuanto a la relación entre proposiciones prescriptivas y normativas, Lyotard escribe: *Al dictar la ley, X decreta que debe respetarla. Al respetarla, Y dicta la ley de nuevo. Sus nombres Y y X son en principio perfectamente intercambiables por lo menos en las dos instancias, el destinador de la proposición normativa y el destinatario de la prescriptiva*¹⁴. Es la legitimación de lo ordenado por el derecho, por la responsabilidad ética.

Lawrence Lessig es un profesor y abogado constitucionalista. Lessig escribió su libro "El código y otras leyes del ciberespacio", para fundamentar la tesis de que el software es ley, pero una ley no legislada ni discutida por el derecho, ni legitimada por la ética. Para Lessig:

*En el espacio real somos capaces de reconocer la manera en que las leyes regulan -por medio de las constituciones, los estatutos y otros códigos legales-. En el ciberespacio debemos llegar a comprender cómo el código regula -la manera en que el hardware y el software, que hacen del ciberespacio lo que es, regulan el ciberespacio tal como es-. El código es la "ley" del ciberespacio. ... entonces, "el control del código es poder". Mitchell escribe: "Para los ciudadanos del ciberespacio, ...el código... se está convirtiendo en un elemento crucial dentro de la batalla política. ¿Quién escribirá el software que estructurará en una medida cada vez mayor nuestras vidas cotidianas?". En el mundo actual, los autores de código son, cada vez más, creadores de leyes, es decir, legisladores. Ellos son los que establecen la naturaleza del ciberespacio. Sus decisiones, llevadas a cabo actualmente en los intersticios de las formas de codificación de la Red, definen lo que ésta es.*¹⁵

¹³ Lyotard, 1999, p. 65.

¹⁴ Lyotard, 1999, p. 118.

¹⁵ Lessig, 2001, p. 25 y p. 118

Independientemente de que los programadores sean legisladores *de facto*, desde la ética hay otros elementos más importantes.

Primero, cuando recibimos una orden, decidimos de acuerdo con nuestra conciencia si la obedecemos o no. Es nuestro privilegio decidir (y asumir las consecuencias). En el caso del software, las órdenes se obedecen. Punto. Aquí no hay libre albedrío. Hay ejecución de una sentencia lingüística por un mecanismo, sin intervención humana.

Segundo, las órdenes se oyen, y son interpretadas por una persona. En el software, las computadoras no interpretan nada. No tienen conciencia. Son máquinas. Se hace, sin oír.

Tercero, al no existir la posibilidad de aplicar la razón práctica (la voluntad pura) ante una obligación¹⁶, la ejecución del software da origen a acciones no éticas, aéticas o paraéticas.

Cuarto, en la operación de los programas de cómputo, al haber obligación sin implicación¹⁷, se anula la responsabilidad ética de los usuarios de un programa. No hay comunidad ética alrededor de la ejecución de sentencias lingüísticas de un programa de cómputo.

1.3. Funciones y Programas

Los párrafos se construyen concatenando proposiciones¹⁸, de la misma manera en que las funciones se construyen concatenando líneas de código, y los programas se construyen concatenando funciones. Pero un párrafo -al igual que una función de un programa- es relativo a un género del discurso.

Entonces necesitamos saber que es un *Juego de lenguaje*. La definición original es de Wittgenstein: "El todo formado por el lenguaje y las acciones con las que está entretejido"¹⁹, asimismo, los juegos del lenguaje son innumerables²⁰. En la traducción española de Jean-François Lyotard se utilizan indistintamente en su lugar los términos "géneros del lenguaje", "géneros del discurso" y "régimen de proposiciones". Para este filósofo: "Dos proposiciones de régimen heterogéneo no son traducibles la una a la otra"²¹.

Hay incontables géneros del discurso, y en muchos de ellos no interviene actualmente el software. Aparte de los problemas discutidos en el apartado anterior del software con el discurso ético, los diversos géneros de discurso cognitivos, científicos o filosóficos buscan entender, explicar. El software está en un nivel operativo, funcional, no en un nivel especulativo o causal.

Además, muchos géneros de discurso son dialógicos: operan en el plano de la comunicación humana. El software no dialoga, porque no entiende nada de lo que hace.

Sin embargo, el software penetra fácilmente muchas áreas de la actividad humana, donde se utiliza el género de discurso técnico (que busca maximizar el resultado y el

¹⁶ Lyotard, 1999, p. 143

¹⁷ Lyotard, 1999, p. 149

¹⁸ Lyotard, 1999, p. 54

¹⁹ Wittgenstein, 1958, p. 25

²⁰ Wittgenstein, 1958, p. 39

²¹ Lyotard, 1999, p. 10

rendimiento)²², y aquí hablamos de los géneros de discurso económico, financiero, administrativo, fiscal, contable, mercadológico, industrial y comercial (para todas las ramas de la industria y todos los giros comerciales).

1.4. Software

Entonces, el software no es un nuevo género de discurso, toda vez que puede representar proposiciones provenientes de una multitud de géneros de discurso. El software es un lenguaje, porque “en el fondo, la definición de una lengua es que se puede traducir a otra”²³, y se pueden traducir proposiciones lingüísticas de cualquier lenguaje natural al software, y viceversa. Pero el software es una tecnología nueva y desequilibrante, que perfectamente podría estar en el núcleo de la revolución tecn-económica que vivimos y que denominamos “globalización”. Para finalizar, dos puntos: por un lado hay dos verbos que están en la estructura misma del desarrollo actual de software, y por otro lado discutiremos las relaciones entre pensamientos, libros y software.

1.4.1. Ser y tener: Herencia y agregación

Ahora bien, hay dos verbos que tienen un uso muy extendido en el software: ser y tener. El lingüista español José Luis Herrero ubica a los verbos ser y tener (junto a hacer y dar), como “verbos soporte”. Para Herrero:

El verbo soporte se encarga de actualizar sustantivos predicativos. Pero no sólo da (“sirve de soporte para”) informaciones de número, tiempo y persona, sino también de la naturaleza interna del desarrollo de la acción o del acontecimiento (aspectuales). Está más o menos vacío de contenido semántico (referencial). Es quizá uno de los rasgos más originales del lenguaje natural (frente a los lenguajes artificiales o a otras expresiones humanas). Es producto de un proceso, probablemente largo y no bien conocido todavía, de gramaticalización imperfecta de unidades léxicas llenas²⁴.

El verbo *ser* es un verbo auxiliar, y en el software se usa como definición de la herencia de clases. La herencia de clases es un concepto definitorio de la programación orientada a objetos, y es implementada en todos los lenguajes de programación orientada a objetos. En inglés, la herencia se conoce como relación *is-a*. Siempre que se puede aplicar la sentencia “es un” entre dos clases, se debe codificar la relación de herencia entre ambas clases. Es difícil exagerar la importancia de la herencia: casi toda la programación que se hace hoy día, utiliza ampliamente el concepto de herencia.

El verbo *tener* es un verbo transitivo. El verbo *tener* da sustento a la relación de agregación entre clases. Cuando entre dos clases se puede afirmar que una “tiene” o “contiene” a la otra, o que una “es parte” de la otra, estamos ante una agregación entre dos clases. Todos los lenguajes orientados a objetos (es decir, los lenguajes de programación más usados en la actualidad, soportan la relación de agregación.

Así, los verbos ser y tener forman parte de la estructura de los lenguajes de programación modernos. Su semántica está empotrada en la estructura misma de la programación actual. Para Grady Booch (tal vez el científico más reconocido del paradigma de programación orientado a objetos), la forma canónica de un sistema complejo se crea a partir de la estructura de objetos (construida a partir de las

²² Lyotard, 1999, p. 164

²³ Lyotard, 1992, p. 69

²⁴ Herrero, 2001, p. 456.

relaciones "tiene-un"), más la estructura de clases (o de herencia, construida a partir de las relaciones "es-un")²⁵.

1.4.2. *Pensamientos, libros y software*

Hay una gran diferencia entre los pensamientos de una persona, y las sentencias lingüísticas codificadas en un software o escritas en un libro. Para Lyotard:

Los pensamientos son nubes. Los pensamientos están a merced de empujones y tirones a velocidades variables. Son profundos, aunque el centro y la superficie son de la misma clase. Los pensamientos nunca dejan de cambiar su lugar unos con otros. Cuando piensas que has penetrado profundamente en su intimidad al analizar su llamada estructura o genealogía o incluso su posestructura, es en realidad demasiado tarde o demasiado pronto. Una nube proyecta su sombra sobre otra, la forma de las nubes varía según el ángulo desde el que nos acerquemos a ellas²⁶.

En la mente humana los pensamientos cambian, se adaptan y se funden entre sí. Por el contrario, el software contiene pensamientos, pero son pensamientos estáticos, que se ejecutan una y otra vez, hasta que son modificados por el mantenimiento del software hecho por un programador.

Pensemos en las diferencias y semejanzas entre los libros y el software. Tanto los libros como el software contienen pensamientos, y dichos pensamientos son "estáticos", están "fijos". Un pensamiento sólo puede cambiar en un libro si el autor del libro cambia el pensamiento en una nueva edición. Un pensamiento sólo puede cambiar en un software si un programador cambia dicho pensamiento en una nueva versión. Pero los libros abarcan todos los ámbitos del conocimiento humano, los libros se introducen a todos los géneros de discurso, mientras que el software se introduce sólo (por ahora) en algunos géneros de discurso, especialmente (pero no solamente) los que son relevantes desde el ámbito de la productividad y la eficiencia económica.

Otra diferencia es que los libros se escriben en lenguaje natural, mientras que el software se escribe en lenguajes de programación, como C++, Java o Visual Basic. Al momento de traducir sentencias del lenguaje natural a un lenguaje de programación, se "oculta" la sentencia lingüística original. Además, los lenguajes de programación en la actualidad están influidos por la concepción histórica de que los programas de cómputo eran constructos matemáticos, no construcciones lingüísticas, y entonces, los lenguajes de programación están diseñados para que la traducción de sentencias matemáticas sea una tarea trivial, mientras que en la actualidad es menos directa la traducción de sentencias lingüísticas a los lenguajes de programación. Pero se puede hacer.

Y finalmente, la diferencia más fundamental entre libros y software: los libros están hechos para que los lea una persona, mientras que el software está hecho para su ejecución automática. El software sería una especie de "libro ejecutable", construido por personas, pero ejecutado por máquinas.

²⁵ Booch, 1996, pp. 14-16.

²⁶ Lyotard, 1992, pp. 18-19.

Bibliografía

- Booch, Grady. *Análisis y Diseño Orientado a Objetos con aplicaciones*. Segunda Edición. Addison Wesley, USA, 1996. 642p.
- Educ.ar. *Relación entre léxico y sintaxis*. Disponible en el portal educativo del estado argentino:
http://aportes.educ.ar/lengua/popup/relacion_entre_lexico_y_sintax.php, 2006.
- Fernández-López, J. *Clasificación sintáctica de los verbos*. Disponible en <http://culturitalia.uibk.ac.at/hispanoteca/Grammatik-Stichworte/Gram%C3%A1tica%20espa%C3%B1ola/Verbos%20-%20Clasificaci%C3%B3n%20sint%C3%A1ctica.htm>, 2006.
- Herrero, José Luis. *Los verbos soportes: ¿gramática o léxico?*, en F. Sánchez Miret, Actas del XXIII Congreso Internacional de Lingüística y Filología Románica, Salamanca, 24-30, Septiembre, 2001, ed. Niemeyer, vol. II, pp. 453-467.
- Lessig, Lawrence. *El código y otras leyes del ciberespacio*. Taurus, Madrid, 2001. 540p.
- Lyotard, Jean-François. *La Diferencia*. Gedisa, Barcelona, 1999. 224p.
- Lyotard, Jean-François. *Peregrinaciones*. Cátedra, Madrid, 1992. 160p.
- Míquelez, Luis Vicente. *Identidad y lazo social*. Disponible en http://reunionesdelabiblioteca.com/conferencia_8_agosto_miguelez.htm, 2001.
- Wittgenstein, Ludwig. *Investigaciones filosóficas*. UNAM, México, 2003. De la primera edición (póstuma) en alemán, de 1958. 550p.
- Yngve, Victor H. "Our Double Anniversary", *Proceedings of the 20th annual meeting on Association for Computational Linguistics*, Vol 20, 1982, p. 92.

SERGIO ELLERBRACKE ROMÁN

sergio.ellerbracke@univa.mx

Profesor e Investigador de la Universidad del Valle de Atemajac y estudiante del Doctorado en Sociedad de la Información de la Universitat Oberta de Catalunya
Av. Tepeyac 4800, Fracc. Prados Tepeyac, Zapopan, Jalisco, México, 45050