WEBCOMS: TRANSACTIONS AS OBJECT-FLOW NETWORKS FOR THE WOS

SIMON SCHUBIGER, OLIVER KRONE, BÉAT HIRSBRUNNER
Chemin du Musée 3
1700 Fribourg
Switzerland
{simon.schubiger, beat.hirsbrunner}@unifr.ch
oliver.krone@swisscom.com
Phone: ++41 26 300 84 68, Fax: ++41 26 300 97 31

Abstract
Data-flow networks - or object flow networks in their more general form - are a suitable co-
ordination language for distributed applications. Their advantages include easy to understand
 diagrams, natural parallelism and fault tolerance. Object-flow networks are used for coordina-
tion at user level in the context of the WOS. In addition, the implementation is based on WOS
services and resources. A basic framework based on attribute schemes, attributes, WEBCOMS
(Web Components) and channels is introduced. WOS specific questions such as usage of ware-
houses, versioning, service and resource lookup will be addressed as well. As this paper reflects
the state of an ongoing project, an outlook presents the future directions.

Keywords: Web Operating System, Flow Based Programming, Coordination, Distributed Sys-
tems, Concurrency, Communication Models.

1 Introduction

Data flow networks are widely recognized as a suitable coordination language for distributed
applications. Their advantages that is easy to understand diagrams, natural parallelism [5]
and fault tolerance [3] are the result of research being conducted in different areas such as
programming languages [16] [2], coordination languages [4] and programming paradigms [12].

Data-flow networks have several advantages in a distributed context:

- Data-flow networks exhibit a natural parallelism. Because the components only depend on
  their inputs, every component can be executed independently of the others. The structure
  of the network maps well to today’s network of workstations that physically have a similar
  structure.

- Data-flow networks can be made fault-tolerant. Upon failure of either a channel or a
  component, the network can be automatically reconfigured to use alternative components
  and channels.

- Data-flow networks have a wide coverage in the literature. Entire languages and systems
  are build on the data-flow paradigm [12] such as Lucid [16], GLU, [5], Strand [2] just to
  mention a few.
Data-flow networks are used successfully in many real-world settings. Applications range from hardware (super scalar/pipelined CPU's) to tools (development environments) to business applications (online transaction systems).

Using a data flow network in the context of the WOS (Web Operating System [8][9]) is a suitable choice since the WOS requires a resource presentation and control interaction going beyond the current desktop systems [14]. In addition, the WOS provides distributed computing through its services which map directly to the nodes of a data flow network. Although the WOSRP (WOS Request Protocol [7]) provides a basic mean for formulating requests, a coordination mechanism is desirable in the context of the WOS. Formulating requests which include several sub-requests and have specific constrains requires an additional layer. Coordination on behalf of data-flow networks was partially implemented in the WebRes project [14] and the results suggested to continue in that direction.

The rest of this paper is organized as follows: in section 2 we place object flow networks in the context of the WOS, the attributes schemes and objects are introduced in section 3. WebComs and channels are presented in section 4. Section 5 is devoted to a description of object-flow networks as the composition of WebComs and channels and an outlook in section 6 concludes this paper.

2 WOS and Object Flow Networks

![Diagram of WOS and WebComs]

Figure 1: Information about WebComs is stored in WOS warehouses. This information is used to build the WebCom-layer as indicated by the lines from the warehouses to the WebComs. Interconnected WebComs form a network of encapsulated WOS services and resources. This is shown by the mapping between WebComs and WOS services and resources.

An Object-flow network as presented in this paper is a graph with a set of vertices called WebComs or plugs and a set of edges called channels. Channels are created through matching

---

1Graphically built networks are easy to understand for the user and have a direct mapping to WebRes resources and runtime linking [6].
of compatible plugs in the WEBCOM space. WEBCOMs come in two flavors: active and passive. A passive WEBCOM is specified by its attribute scheme only. Documents are typical passive WEBCOMs. Every change in some set of attributes results in an object flow through the network. This data is processed by active WEBCOMs which have an action in addition to their attribute schemes.

Figure 1 depicts the relations between the WOS and WEBCOMs for a compilation scenario. Two active WEBCOMs are shown, called compile and link which are connected to passive WEBCOMs (files) represented by WOS resources hello.c, hello.o, and hello respectively. The active WEBCOMs in this example are mapped to the WOS resources gcc and ld.

3 Attributes Schemes and Objects

Because WEBCOMs have real world representations such as programs and files, they represent some known concepts. These concepts are captured in attribute schemes. In this respect attribute schemes are similar to intentions in the intentional programming style [1][15]. Attribute schemes serve as abstractions for these concepts. New attribute schemes are constructed by a description or by a composition using set operations. For example the attribute scheme of hello.c might be defined as c-file:

\[
\text{scheme c-file is file union unix-file union c-source;}
\]

\[
\text{scheme file is }
\]
\[
\quad \text{name(}\text{Name}\text{)} :\text{ volatile, persistent, native(}\text{name}, \text{Name}\text{)}.
\]
\[
\quad \text{length(}\text{Length}\text{)} :\text{ volatile, persistent, length(}\text{length}, \text{Length}\text{)}.
\]
\[
\quad \text{path(}\text{Path}\text{)} :\text{ volatile, persistent, path(}\text{path}, \text{Path}\text{)}.
\]
\[
\quad \text{content(}\text{Content}\text{)} :\text{ volatile, persistent, content(}\text{Content}\text{)}.
\]
\[
\text{scheme c-source is }
\]
\[
\quad \text{content(}\text{Content}\text{)}.
\]
\[
\quad \text{semantic(}\text{Semantic}\text{)} :\text{ implicit, dependsOn(content)};
\]
\[
\text{scheme unix-file is }
\]
\[
\quad \text{i-node(}\text{Inode}\text{)} :\text{ volatile, final, native(}\text{i-node}, \text{Inode}\text{)}.
\]

The definition of an attribute scheme is either implicit such as in c-source.semantic(Semantic), defined by a platform dependent (native) implementation like unix-file.i-node(Inode) or a relation based on other attributes as in c-source.semantic(Semantic). Various relationships between attributes can be stated as well. For example one attribute might be just a synonym for another attribute or the value of an attribute depends on the value of another as in semantic(Semantic) :- dependsOn(content). Having attribute schemes without a definition makes sense because certain relationships between attribute schemes can be established without referring to a definition. For example what one normally expect from a C-compiler is,

\[\text{2}\text{The notion of intentional programming is also used with a somewhat different meaning in the context of Lucid [13].}\]
that the object file has the same semantics as the source file. A C-compiler might then claim
that the semantic(Semantic) attribute scheme is invariant under compilation. This property
of a C-compiler can be stated without saying what semantic(Semantic) actually means. Such
an attribute scheme is said to be implicit because there is an implicit agreement on its meaning.
Instances of attribute schemes are called objects and their parts are called attributes. The
hello.c object might be

    object hello is a c-file {
        name(hello.c);
        length(3456);
        ...
    }

Upon a change of one or more attributes, a new instance of an attribute scheme is created,
called an object or a version, inheriting the unchanged attributes from the previous version.
The system maintains the resulting version history completely or partially depending on the
versioning policy. Such an attribute change is called a transaction which usually causes several
subsequent transactions by the initiated data flow. An example of a transaction is an editor
that saves a new version of hello.c. This transaction will affect several attributes such as
content, length etc. This may trigger some actions like recompilation, which again cause new
transactions changing attributes in object files.

The mapping between attribute schemes and the entities will be stored in WOS warehouses
as well as low-level platform specific information and higher level concepts such as files.

4 WEBCOMs and Channels

As already mentioned in the introduction, we use passive and active WEBCOMs.

Passive WEBCOMs

Passive WEBCOMs are representations for entities which can not be executed by some means.
Documents of any kind are typical examples of passive WEBCOMs. Given a set W of WEBCOMs,
a passive WEBCOM w' \in W has an attribute scheme w'.A. Usually the system will infer the
attribute scheme of a passive WEBCOM. For example the system can use the file extension or
the content as a hint for finding the attribute scheme. Some attribute schemes may be defined
directly by the entity itself through mechanism like XML metatags [10]. Other attribute schemes
may be added from the context such as file system specific information, character encoding,
access costs and so on.

Active WEBCOMs

Active WEBCOMs are representations for entities which can be executed by some means. Prog-


Figure 2: A C-compiler as an active WEBCOM. (x) action, (A) attribute schemes, (i) input plugs, (o) output plugs.

Channels

A channel is the edge in an object-flow network $G$ and connects two WEBCOMs. We distinguish two cases of connectivity: either between two active WEBCOMs or between a passive WEBCOM and an active WEBCOM. Two passive WEBCOMs can never be connected.

For two active WEBCOMs the following connectivity relation is defined: an input $w_c.x.i_α \in w_c.x.I$ can be connected to an output $w_\phi.x.o_β \in w_\phi.x.O$ thus forming a channel if they share the same attribute schemes:

$$w_c.x.i_α.A = w_\phi.x.o_β.A$$  \hspace{1cm} (1)

For an active and passive WEBCOM we define that a plug $w_c.x.p_α$ of $w_c$ can be connected to a WEBCOM $w_\phi \in W$ if the attribute scheme of the plug is a subset of the WEBCOM's attribute scheme:

$$w_c.x.p_α.A \subseteq w_\phi.A$$  \hspace{1cm} (2)

A plug $w_c.x.p_α$ of $w_c$ is said to be connected if either (1) or (2) holds. A pair of elements satisfying the relation “to be connected to” is called a channel.

An action $w_c.x$ is said to be totally connected when $\forall w_c.x.i \in w_c.x.I, \exists w_\phi \in W$ such that:

$$\begin{cases} 
\text{either a } w_\phi.x.o \text{ is connected to } w_c.x.i \\
\text{or } w_\phi \text{ is connected to } w_c.x.i
\end{cases}$$  \hspace{1cm} (3)

The input attribute scheme can be seen as the precondition of the action. The output attribute scheme together with the inter-attribute scheme relationships as the postcondition of the action. Thus the description of an active WEBCOM acts as a software contract [11] between the user of the WEBCOM and the WEBCOM.

47
Relations

A relation is a function $A_1 \times A_2 \times \ldots \rightarrow \{\text{true, false}\}$ of two or more attributes that evaluates to true or false as soon as there is sufficient information available to evaluate the function. A relation is said to hold if it evaluates to true. If a relation evaluates to false, actions have to be found that transform the attributes in a way that the relation again evaluates to true. Starting from the attributes involved in the relation, plugs and finally the actions are found.

5 Object-Flow Networks

An object-flow network is a generalization of the data-flow network in the sense that not only passive data items are allowed to flow through the network but also associated code. Having code together with data enables the system to use lazy evaluation techniques as well as remote access for attributes. The network consists of nodes (WEBComs and plugs) and edges (channels). Objects are dynamically constructed upon a change in some attribute set and flow through the channels. Objects may be either associated with a data flow through some dynamic constructed entity such as a pipe or a TCP connection or may have an observable representation such as an intermediate file and are represented by passive WEBComs in that case. A WOS warehouse might also serve as object store for channels that can not be represented by other means.

Figure 3 shows an example of an object flow which compiles and links the file hello.c

![Diagram showing an example of an object flow which compiles and links the file hello.c.]

Figure 3: An example network which compiles and links the file hello.c.

In this example a user specifies as input a passive WEBCom, a file (hello.c), with a set of appropriate attributes, and a goal WEBCom, the compiled file hello. Since the attribute scheme of the plug of the compiler is a subset of the attribute scheme of the file, condition (2) is fulfilled and the two WEBComs will be connected which results in an activation of the compiler. The compiler’s output plug and the linker’s input plug share the same attribute schemes, condition (1) is fulfilled and therefore the two plugs match. The final file is created as the result of the matching of the attribute scheme of the output plugs of the linker with the attribute scheme of the
destination file, here, again condition (2) holds. Note the implicit attribute semantic(Semantic) which must not be changed during the flow of WebCOMs through the system.

The beauty of this approach lies in its potential. For reasons of simplicity we chose a classical data-flow example which can easily be implemented using UNIX's make command. However, the generalization of this approach, especially when used in context of the WOS, opens the door for new applications. A typical application area would be unified messaging. Imagine WebCOMs which transform the contents of a mail-message into different media, depending on the needs of a user: a WebCom with the original message and a WebCom with an attribute scheme reflecting the intended transformation are given to the system. The search and lookup procedures to find the appropriate resources to transform the message will be transparently handled by the system, that is resources scattered all over the Web can be used to perform the transformation. Especially the implicit attribute approach is very well suited for this type of application area, since the message content is invariant under transformation.

6 Outlook

The system currently under development will continue the work of WEBRES [14]. The idea of WEBRES was to combine coordination theory with Web technology and use the Web as an interactive meeting point for resource sharing. At interface level, the user of our environment accesses available resources such as files, CPUs, and the like or whole applications via typical operations known from classical desktops. The visibility of resources on the Web can dynamically be extended or reduced, therefore enabling collaboration functionality by changing the accessibility of formerly local resources at a system-wide, global level.

By extending WEBRES, a graphical network builder will be the first concrete syntax of the still to define network language. This tool will allow to interactively build networks by explicitly mentioning all the required WebCOMs and channels. It is planned to use the system right from the start during the development process of the system itself. System composition, compilation, version management and documentation will become automated through appropriate networks.

In a second phase the network builder will be extended to include an interactive query mechanism. The goal of this second phase is to let the user formulate its software requirements without explicitly mentioning the WebCOMs as it was necessary in the former case. In a dialog with the user the system will make proposals to the user to fulfill the software requirements. This requires the system to infer the WebCOMs and presenting them to the user which finally can build the network. This phase will use agent technology for finding and negotiating suitable WebCOMs on the Web.

The last phase will automate the entire construction process. By accumulating knowledge about the WebCOMs available, the system infers the network itself based on a minimal set of user hints. The user should be able to state its goal in a few short phrases in a language understood by the system. This language maybe even a natural language. Based on the systems knowledge and a user profile, the system will produce the optimal network to reach the goal.

References


