ABSTRACT
Over the past few years, traditional software products, sales and licensing schemes have declined, whereas business value and revenues have shifted to SaaS-based schemes. Even so, most research has focused primarily on the technical layer (i.e. service invocation, integration, coordination, etc.). As a result, most SOA solutions available on market still do not feature a service “face” for human users. The SOA front-end of those that do is based on monolithic, rigid and non-customizable user interfaces and portals that invoke back-end services and processes in an ad-hoc manner as needed. This paper presents the rationale behind a novel user-centered visual service composition system being developed by the European FP7 FAST Project consortium. This service composition system aims to enable service composition by guiding non-technical users through an open innovation process. The proposal formally models the component model, techniques and languages. Also it leverages some well-known Web 2.0 principles in order to bridge the gap between the service technical layer of a SOA and its end users. This should improve user appreciation of the benefits of such a system, enabling them to easily mash up their own service front-end from its basic and/or available building blocks.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems – Distributed applications;
D.1.7 [Programming Techniques]: Visual Programming;
D.2.2 [Software Engineering]: Design Tools and Techniques – Computer-aided software engineering (CASE), Flow Charts, Modules and Interfaces, Top-down programming, User-Interfaces;
H.4.m [Information Systems]: Information Systems Applications – Miscellaneous;
H.5.2 [Information Interfaces and Presentation]: User Interfaces – GUI, Screen design, User-centered design

General Terms
Design, Languages

Keywords
User-centered service oriented architectures, service front-ends, composite applications, component models, composition languages, composition techniques

1. INTRODUCTION
The changes in IT evolution and the software business in recent years have significant implications for software products, developments and use. Over the past few years, traditional software products, sales and license fees have declined, whereas business value and revenues have shifted to SaaS-based services[2, 9]. SaaS (Software as a Service) is a recognized approach that emerged from the traditional ASP (Application Service Provider) delivery method. As a result, Internet services are becoming more important than product revenues. Since the year 2000, enterprises and private customers are refusing to pay for standardized or commodity-type software products, and both the B2B and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

uWAS09 ’09 Kuala Lumpur, Malaysia
Copyright 2009 ACM X-XXXXX-XX-X/XX/XX ...$10.00.
B2C IT economies are now based on Service-Oriented Architectures (SOA). SOAs increase asset reuse, reduce integration expenses and improve the rate at which businesses can respond to new demands. Web 2.0 phenomena \[23\] go a step further and have led to a focus on service front-ends in order to bridge the gap between the service technical layer and end users. There are key proposals giving ‘DIY’ guidance on evolving SOAs to meet end-user demands and requirements, like iGoogle\[4\], Yahoo! Pipes\[3\], ServFace\[2\] or EzWeb\[5\]. These solutions promote the adoption of SOA front-ends. SOA front-ends empower end users and exploit context and knowledge. Their aim is to get end users to appreciate the benefits of SOA by fostering composition, loose coupling and reuse on the front-end layer, thus reaching a user-centered service conception.

The objective of this movement is that any enterprise or end user should be able to create their own software solution that exactly meets their requirements within a very short time-to-market by composing a new complex solution built from heterogeneous resources and their front-ends. However, existing SOA front-ends are still based on monolithic and non-customizable user interfaces and portals that invoke back-end services and processes in an ad-hoc manner as needed. Most services do not feature a face for human users, as they reside on no more than a technical layer. Current approach for creating service interfaces relays on heavy-weight engineering skills far from the end user ones. This gap can be reduced by the use of reusable and composable interface-enabled components. For this reason, the challenge is to come up with a visual service composition framework to enable the composition of user-centered services guiding non-technical users during the open innovation process. A visual composition framework is the simplest solution for bringing composition processes closer to the Long Tail of Internet \[4\]. Using the framework, for example, services and resources could be composed and interconnected by a simple drag-and-drop of components, services and operators from a palette or catalog.

Existing approaches such as Internet mashups, while valid in the small situational applications, are unsustainable for complex composite applications \[10\], and other approaches like BPEL are unsuitable for dealing with non-specialized end users. The main issue is that service front-ends are constructed in an ad-hoc manner and without user-oriented tools allowing visual composition of services and their associated user interfaces. When provided, these tools speed up the time-to-market while lowering the cost, thus allowing to cover a larger share of the long tail.

The restricted functionality of most solutions’ UIs is a major shortcoming of the Future Internet. They tend to be limited to the combination of just content rather than applications. High dependency on the underlying computing infrastructure is another limiting factor. Also, changes in the original wrapped applications, the back-end services or infrastructure is another limiting factor. Also, changes in applications. High dependency on the underlying computing render user-service interaction unusable.

Formally modeling the component model, the composition techniques and the composition language for visual services composition will standardize the composition process. It will homogenize this process to provide a more formal composition guideline, fostering the sustainability, performance and reliability of applications built ad hoc.

The formal modeling of the compositional process depicted in this paper is focused on the rationale behind FAST\[5\], a research project that aims to create a complex gadget development environment. FAST will empower end users to co-produce and share instant composite applications and their components \[22\].

The remainder of the paper is structured as follows. First we depict the generic requirements for software compositional systems and several advisable properties on Section 2. Then, such requirements are particularized on the user-centered compositional applications development domain, creating specific ones and therefore generating the proposed composition model in Section 3. Next, we analyze the proposed composition model into its main parts: Section 4 deals with the component model, Section 5 with the composition technique, and Section 6 with the composition languages. Finally, Section 7 presents other related work and Section 8 concludes this paper and presents a brief outline of future work.

2. GENERIC REQUIREMENTS FOR COMPOSITIONAL SYSTEMS

As stated many times in literature \[6, 30\], three key aspects need to be defined in order to describe a software composition system or, more generally, design a component-based software architecture: a component model, a composition technique and a composition language, as shown in Figure 1. The following subsections discuss each of these aspects.

![Figure 1: Graphical representation of the compositional meta-model.](http://www.fast-project.eu)

2.1 Component Model

The component model defines what the different elements of the composition model should behave and relate each other. As stated in \[6\], every included element in the component

---

\[1\]Google: Google Personalized Homepage [http://www.google.com/ig](http://www.google.com/ig) (last checked on 2009-07-15)


\[4\]Morfeo EzWeb [http://ezweb.morfeo-project.org/](http://ezweb.morfeo-project.org/) (last checked on 2009-07-15)

\[5\]FAST Project, [http://www.fast-project.eu](http://www.fast-project.eu) (last checked on 2009-07-15)
model should fulfill several requirements to create a useful composition system:

- **Modularity.** If the architecture is meant to produce reusable pieces of software, which is the key feature of a component-based architecture, all the components defined in the component model, and more specifically their interfaces, must be designed to be reusable. Thus, a component-based architecture can shorten the time-to-market, thereby reducing the production costs.

- **Parameterizability.** To make the most of component reuse, components should be able to be parameterized using generic data to tailor them to different similar purposes.

- **Standard interfaces.** Taking into account that component functionality is encapsulate and interfaces are their only visible part, standardized interfaces are especially valuable for composition purposes, enhancing reusability and shortening the learning curve.

### 2.2 Composition technique

Another requirement for creating a composition model is to define its composition technique. A composition technique states how and determines the available mechanisms to compose the model elements. Furthermore, mediation between components, in order to get them connected, must be defined within this task. To deal with these issues, any composition technique should have the following features:

- **Connection.** Obviously, every component in the model should be able to connect to other components. Thus, it is necessary to adapt the component parameters, protocols and assertions. The composition technique must be aware of this issue and provide adaptation and gluing. Adaptation stands for the process of transforming the component to fit into an interface. The gluing process deals with mediation between components. Both processes eventually increase component reuse.

- **Extensibility.** The designed technique should account for the possibility of automatically extending existing systems with new functionality and non-functional features without changing the existing system components.

- **Aspect Separation.** It is important to think about components not as black boxes, but also as covering functional and non-functional features.

- **Scalability.** Compositions should scale in binding time and technique. Increasing the number of components or type of components are likely and predictable, therefore the composition technique should scale properly.

- **Metamodeling.** The composition technique needs to have a model of the components if they are to be adapted and transformed.

### 2.3 Composition language

Last, but not least, it is necessary to define a composition language. A composition language determines how composite systems are specified. It must define how to describe the architecture of any system conforming to the defined composition model. Thus, it must be completely aligned with the component model (i.e. it must define how the components are represented) and the composition system. Any composition language must meet the following requirements:

- **Product-Consistency Support.** Any composition language must deal with product-consistency software to help, as a result of its design, to assure quality features.

- **Software-Process Support.** The resulting language must be powerful and expressive enough to support any composition-based software design process. This includes its ability to express variants and versions of product lines, represent large systems and be easy to understand.

- **Metacomposition Support.** The language itself should be based on the composition process to allow higher-level composition systems.

### 3. COMPOSITION MODEL FOR USER-CENTERED SERVICE-BASED APPLICATIONS

In the previous section, we defined the generic requirements for defining compositional systems in the shape of a component model, a composition technique and a composition language. In this section, those requirements will be instantiated taking into account the additional requirements and constraints taken from the application domain.

These requirements are derived from the type of the problem to be solved: enabling user-centered composition of applications from services and other back-end resources. Current mashup tools are not suitable for completely solving the problem as identified in [3][15] and taken into account by reference service architectures such as NEXOF-RA [15] from the European Technology Platform NESSI. This work is aligned with the vision of both NESSI and the Open Alliance on Service Front-Ends in the sense of offering user-centered mechanisms for user-centered creation of service front-ends.

Since we aim to create service-based composite applications, a brief study of the different aspects that such applications tackle is following. It is remarkable that all of these aspects must be implemented by the components of the component model.

- **Service discovery.** When building compositional applications there is a need for component discovery in order to gather all the required pieces: the actual services, and adaptation and integration resources. To do this for the specific case of web services, several approaches based on metadata and a global catalog or register have been proposed. Some of them have not succeeded due to poor semantics [10] or failure to gain a critical mass [11].

- **Service invocation mechanism.** At some point, services must be invoked by the composite application. The challenge related to this aspect is to put its inputs in terms of the respective service and then translate back the results to provide the composite application dataflow. This is an issue that cannot be overlooked since it has an impact on the ability to combine services from diverse sources or even quite different services. Some applicable techniques are semantic web
As our contribution to the European FP7 project *Fast and Advanced Storyboard Tools (FAST)* [17], we are putting into practice the previous ideas into the FAST Composition Model, whose details are explained in the next sections. Specifically, Section 4 deals with the FAST Component Model, Section 5 explains the composition technique, and Section 6 presents the needed composition languages.

### 4. FAST COMPONENT MODEL

The FAST Component Model defines the components and how they are organized into two different levels of composition for the specific requirements of FAST. Figure 2 depicts the FAST components and their relationships which are detailed in the next subsections.

#### 4.1 Resource

In the context of FAST, Web services can be seen as a kind of components that can and should be composed into larger systems [24]. However, Web services are only a particular case of invokable resource to be composed.

One of the main advantages of our proposed wrapping is the fact that the adaptation is not tied to traditional SOAP-based Web Services. We are open to all kind of back-end services, such as databases, legacy systems and even REST-compliant resources. Thus, we are offering a new approach to EAI problems [1].

Bearing this in mind, the component model define a resource as the key component required to wrap or adapt services for subsequent composition. On one side, resources can be seen as an abstraction of an invokable method (i.e. one specific method for a Web service, a POST method for a RESTful service or any other kind back-end resource matching this concept).

On the other side, an standardized interface should allow a composition technique. We propose to model the inputs and outputs of the resources as pre- and post-conditions composed of atomic assertions. They are modeled using semantic technologies and are called facts.
4.2 Operator

Having examined the FAST resources, we now define a subclass of them called operator. Operators are meant to transform and/or modify data in piping processes. No constraints have been placed on operators in FAST, and they are invoked through a common interface as if they were simple-adapted services. FAST defines several instances of general-purpose operators, such as aggregators, filters, selectors or iterators. Moreover, it will be possible to extend them on demand thanks to the use of the designed interface.

4.3 Fact

A fact is defined in the scientific context as an objective and verifiable observation representing assertions regarding a matter. From the FAST standpoint, facts are instances of domain concepts making up the basic information unit of FAST applications. They enable assisted semantic composition which eases the processes the user has to carry out.

To do so, they play the role of pre- or post-conditions for the rest of building blocks. Pre-conditions conform a set of constraints that restricts the execution of components, whereas post-conditions represent changes as a result of the component execution. We will see how pre- and post-conditions play an important role in FAST composition technique in the forthcoming section.

At runtime, facts can also be stored in a knowledge base representing the application state.

4.4 Form

A Form can be seen as a generic graphical user interface acting as a service front-end, responsible for establishing the visual communication and the interaction mechanism with the end user. In our proposed model, forms contain both view and presentation logic (i.e. event management or rendering operations). Forms are considered as black box components so they can be developed in any (Web) technology. Note, however, that they have to be designed as generically as possible to promote their reusability in different application domains.

Taking into account that forms are going to be the interface that end-users will see and interact with, it is important to offer the best user experience. But this is not easy to do by offering just a generic interface. Component parameterization, then, plays a big role here. It is useful for delivering customized forms (for instance, allowing internationalization or adapting general-purpose interfaces into a specific domain...). In some cases, a customized generic form will not meet user requirements either. To solve this problem, the model supports domain-specific forms.

As mentioned previously, we look at forms as black box components. It is necessary, therefore, to define their public interface to support interaction with the other model components. To ensure composition modularity, the forms offer an interface based on the aforementioned pre- and post-conditions.

4.5 Screen

Screens are probably the most important component of our model. They are the minimal functional blocks that can be executed independently. They include both business logic and graphical user interface interconnected with each other. Like the above components, screens have a fact-based interface. This interface will play a key role in their composition to create screenflows, as discussed in the next section. Bearing these constraints in mind, screens can be created in two different ways:

1. linking several resources, operators and a form together in compliance with the FAST composition technique;
2. developing a monolithic and ad-hoc piece of code on the condition that it conforms to the screen interface.

4.6 Screenflow

The last component of our model is the Screenflow. It is FAST's top-level component, and it is literally made up of a set of screens. Specifically, a screenflow is an meaningful aggregation of screens endowed with business logic. The business logic comes from the combination of each screen's inner logic plus the composition logic.

Regarding the flow between screens, it follows a fact driven approach. Both pre- and post-conditions are used to drive the transitions during the screenflow execution. This technique will be explained in section 5.

Finally, note also that screenflows can also be composed with each other in order to create bigger ones. This is possible because they also share the same fact-based interface as the other FAST components.

5. FAST COMPOSITION TECHNIQUE

The aim of the FAST composition technique is to define how the components described in the previous section can be actually composed. This composition is done at two levels:

1. creating screens from existing resources, operators and forms; or
2. composing screenflows from existing screens.

In this section we present the FAST composition technique which handles both processes. It is handle common fact-based interface and the so-called PRE/POST mechanism which will be also presented.

5.1 The PRE/POST Mechanism

Before presenting the two levels of the FAST composition technique, it is important, to convey a full understanding of how it works, to define the way data flows between the different components and how these building blocks interact with each other. The PRE/POST mechanism, based on facts, is how the components get connected to other ones.

In FAST, as discussed earlier, all the components have a common interface whose inputs and outputs are defined in terms of facts. A component can constrain its execution by means of pre-conditions. If a component needs a certain data type in order to execute properly, it will wait until it receives that information. Once the component has received all the data it requires, it executes.

At runtime, components usually generate some output data belonging to a specific data type and a domain class. This is what post-conditions are. Once generated, a post-condition is usually stored into a knowledge base. In screen composition, the post-condition will be propagated along semantic pipes to the next component in the composition chain, whereas in screenflow composition an inference engine will decide the screen(s) the post-condition will be delivered to and eventually, triggering a screen transition.
Both component model and the PRE/POST mechanism contribute to allow the parametrization and adaptation of properties. Components can be parameterized and are modeled by their pre- and post-conditions (Figure 3a). Then, their facts facilitate adaptation by means of concept mediation and class specialization, enriching the original interface characterization of the component (Figure 3b). Finally, facts are connected creating semantic pipes thus allowing gluing (Figure 3c).

Figure 3: Component parameterization and adaptation.

5.2 Screen composition

The lowest level of the FAST composition technique refers to the creation of screens by composing existing resources, operators and forms. We will illustrate how this composition technique works by means of an example login screen depicted in Figure 4.

To get the screen composition started, the end user usually begins by selecting a form that meets her needs (1). Once selected, the form could have to satisfy a pre-condition (2). In that case, the end user will need to look for a back-end resource whose post-condition matches the above pre-condition (3). If the end user is unable to find exactly the proper resource, but there is another one whose output is quite like what she is looking for, she might want to use a filter (an operator) to adapt the data to the specific pre-condition format (4). Back-end resources, operators and the form will be linked by means of semantic pipes. These pipes will guarantee the validity of the data and their data type using semantic matching.

Due to the PRE/POST mechanism, back-end resources have their own pre-conditions and they also have to be satisfied in order to get the screen working. If they can be satisfied by means of other resource post-conditions, the user will connect these resources (5). If not, the unresolved pre-condition will be one of the screen pre-conditions (6), thus having to be solved.

Once the form pre-conditions are satisfied, and depending on the type of form the user has selected (interactive or not), a GUI event could be needed to have the form executed (7). The execution of the form must create some output data (in the shape of a post-condition). Depending on the screen business logic, the post-condition might be propagated to a back-end resource to validate its data (8). Finally, if no errors occurred, the screen post-condition will generate the new fact (9).

5.3 Screenflow composition

The top level of the FAST composition technique is the screenflow composition, which generates a fully functional composite application. As expected, every screen to be composed has a set of attached pre- and post-conditions that will be used to drive the transition from one to another through a set of output facts during the screenflow execution. This way, a screen has two possible states: reachable and unreachable as it is depicted in Figure 5a. If all the pre-conditions of a screen are fulfilled by the facts output during the screenflow execution, the screen will be reachable. Otherwise, it will be unreachable.

Figure 5: FAST screenflow composition process

Note that the end user does not explicitly define any tran-
osition between screens. It is the FAST platform that does this implicitly beforehand, thanks to the PRE/POST mechanism. However, a user that wants to set a fixed transition between two screens can do so by using the Optional Connector flow control to set up the transition.

Due to the PRE/POST mechanism, there would be no obstacle to adding a screen whose pre-conditions were satisfied by the current facts present in the screenflow.

6. FAST COMPOSITION LANGUAGES

In order to meet all the composition language’s requirements, we propose three different representations or languages with different objectives:

- **FAST Visual Composition Language (FVCL).** A visual language allowing final and power users to compose in an intuitive and productive fashion.
- **FAST Modeling Language (FML).** A markup language as a means of intermediate storing of compositions. This representation is not intended to be used directly by users, indeed it is suitable for processing and transmission.
- **Execution language.** The composite applications might be compiled to executable languages in order to be deployed on different execution platforms.

Following subsections show the FAST Visual Composition Language in detail. However, note that, for reasons of space, both the formalization of the FVCL and the presentation of the rest of the composition languages (FML and Execution languages) will be further developed in a forthcoming paper.

6.1 FAST Visual Composition Language

**FAST Visual Composition Language** is the language for visually composing the different FAST components. There are lots of visual languages in the literature [29, 33, 8, 14], some even for describing how services are composed together [26, 32]. However, none were designed from a user-centric perspective. This is the main reason why we have developed a new language.

One of the main issues to be solved when defining visual languages is how to describe the type of representations that the language uses. This language deals with what is to be represented, how it is to be represented, and how to associate the representation with what it represents [21].

6.1.1 Visual representation of FAST components

Following we describe how the FVCL represents the FAST Component Model by means of their graphical view and their meaning.

**Screen.**

During screenflow design, Screens are represented as rounded rectangles with three areas as depicted in Figures 6a and 6b. The upper area contains a caption while the other ones are mean to contain other components.

Reachable screens are colored green (Figure 6a) whereas unreachable ones are outlined in red (Figure 6b). The upper one contains the output fact, whereas the lower one keeps the set of input facts (pre-conditions).

If we are creating a screen by composing other FAST components, the screen view will be quite similar to the one shown in Figure 4.

**Resources.**

Resources are represented very much the same as screens. In fact, both components have a caption and pre- and post-condition areas but in a different layout as depicted in Figures 6c and 6d. As well as screens, their colour depends on their reachability.

**Operators.**

Visual syntax for a binary operator is depicted in Figure 6e. As can be seen, operators are represented as a diamond divided into a couple of areas. The upper one contains the output fact, whereas the lower one keeps the set of input facts (pre-conditions).

**Facts.**

The syntax of facts (both pre- and post-conditions) is visually defined by a small circle (see Figure 6f). It will be filled in with the initial letter of its associated concept or a more elaborate mnemonic acronym generator. Regarding its satisfiability, if the circle is solid (and green), it means the fact is satisfied, whereas if depicted by just a (red) outline, the fact is not fulfilled.
Reachable Screen
F
A
Postconditions
Postcondition C
Postcondition A
Postcondition B
Postcondition D
Postcondition E
Postcondition G
Postcondition F
A R
S
CCCCCF
A
B
C
D
E
F
G

Forms.

*Forms* are needed just while creating *screens*. Their representation is a rectangle whose background shows a thumbnail of the associated user interface and some *pre-* and *post-conditions*, as depicted in Figure 6.

Flow controls.

At screenflow composition level, FVCL defines three optional flow controls just in case the user wants to set the initial or last screen of the screenflow, or fix a transition between a couple of screens in particular. Figure 6 depicts both the begin and end symbols and Figure 6j shows the optional connector useful for fixing transitions.

However, at screen composition level, we only have one flow control called *semantic pipe*. It is represented by a single arrow as can be seen in Figure 6i. Despite it is very similar to the optional connector, their semantics have nothing to do and they are not used at the same time.

6.2 FVCL Views and Visual Scaling

Despite it would be possible to show all the composition information just in one diagram, it would not be user-friendly at all. Only trivial composite applications could be understood unless partial specific diagrams, namely views, allowing to focus on smaller parts of the composite application. These views should not deal with an arbitrary subset of the application but cohesive subsets as loosely coupled with the rest of the application as possible.

Taking the aforementioned into account, FVCL offers several views depending on whether the user is composing at *screenflow* or at *screen* level.

Screenflow View.

This view shows the set of *screens* which conforms the composite application being developed and other optional flow controls as in Figure 7.

*Screens’ pre-* and *post-conditions* define an implicit screenflow which is restricted by the flow controls. For instance, a begin symbol attached to a login screen will guarantee that the first screen to be executed will be the aforementioned screen.

Screen View.

This view is partitioned into *pre-* and *post-condition* areas and a main area. *Pre-condition* area is placed on the left side and contains facts modeling the inputs for the depicted screen. In a similar fashion, *post-condition* area is on the right side.

The main area can be further divided into three layers keeping different kind of components:

- **Form area.** This area, located at the top of main area, must contain exactly one *form*. This component provide the user interface to the whole *screen*, and therefore is mandatory.

- **Piping area.** Most of the piping is specified at this area since it contains all the operators and semantic pipes. Some of the inbound and outbound pipes are connected with facts of the form, the resources or *pre-* and *post-condition* facts.

- **Resources area.** Located at the bottom of the main area, the resources area holds one or more backend resources. They will be invoked when their *pre-conditions* were satisfied, and as a result, their *post-conditions* can propagate though semantic pipes, eventually triggering additional invocations.

Visual Scaling.

Apart from the described main views there are additional auxiliary views for the sake of the visual scaling. One illustrative example takes place when there are too many facts on a pre- or post-condition area of a screen since there is no limit on the number of facts a screen can have. Due to size constraints, the facts can be stacked and shown to users by means of a menu. This is depicted in Figure 7.

7. RELATED WORK

Companies are beginning to focus on people as the entry point to SOA and, therefore, to composite applications [16]. Thus they need a means to bridge the gap between people and services. It is then that they come up against the traditional shortcomings of composite applications. Consequently, a number of user-centric composite application frameworks are beginning to proliferate. Worthy of note is IBM’s solution, named SOA for people [28]. It focuses on a portal framework acting as a SOA front-end to maximize people’s productivity and collaboration. In addition, SAP has created SOA-People. This working group claims that with portal and collaboration software, an SOA environment can simplify the way people interact. The increasing interest in this approach is indicative of the current importance of user-centric SOA in the business world.

However, existing approaches focus on employing particular Web 2.0-based technologies to deliver a front-end to SOA rather than lending attention to the composition process and component modeling. In this paper, we revisit new services and applications, from their creation to their consumption, through their composition following a user-centric SOA approach.

---

**SOA-PEOPLE, [http://www.soapeople.com](http://www.soapeople.com)** (last checked 2009-07-15)
centered approach based on enterprise composite applications, to create a visual composition model that helps to standardize the composition process and improve application performance and reliability \[20\].

### 7.1 Visual Languages and Composition Formalisms

Visual languages and tools have been successfully used for many different purposes (e.g., programming, user interaction and visualization) \[25\]. Visual languages attempt to provide an effective, graphical, non-linear representation that has been successfully applied to modeling (e.g., UML), parallel computing, laboratory simulation, image processing, workflow description, hypertext design, and even object-oriented programming. Following our approach, software composition is potentially a good application domain for a graph-based, visual notation. However, instead of focusing on typical composition issues regarding how the “spatial” architecture of a software system can be specified in terms of components and connectors, we have focused on describing how services should be composed in “time” \[13\]. Apart from describing the data flow structure of the interaction between different services, we have also included a separate description of their control flow dependencies in the FAST Visual Composition Language \[20\] \[12\].

In the past, many graphical formalisms have also been developed in this area. Here some contributions that have been applied to workflow modeling are listed below:

- State Charts, used in the Mentor project to achieve distributed execution of the various workflow steps
- Petri Nets and variations such as Object Coordination Nets (OCaN)

These formalisms have a natural visual representation, which provides the user with a good overview of the partial order of services invocation.

Nevertheless, when applied to service composition, one of the limitations of a visual language based only on control flow is that there is no visual notation for specifying adaptations between mismatching service interfaces \[5\].

### 7.2 Software Composition Framework

The composition approach discussed in this paper is based on the FAST initiative, a STREP project partially funded under the European Commission’s 7th Framework Programme (INF50-ICT-216048), as part of NESSI \[7\], the Networked European Software and Services Initiative that is an European Technology Platform dedicated to Software and Services. Other similar initiatives are beginning to proliferate in this research field. Of these, the NEXOF-RA project \[8\] deserves a special mention. The authors of this paper collaborate and participate actively in this project, which aims to build the Reference Architecture for the NESSI Open Service Framework by leveraging research in the area of service-based systems, and to consolidate and trigger innovation in service-oriented economies. NESSI is the European Technology Platform dedicated to Software and Services. Its name stands for the Networked European Software and Services Initiative.

ServFace is another STREP project funded under the European Commission’s 7th Framework Programme related partially to the ideas presented in this paper. This initiative aims to add an integrated UI description and development approach to SOA concepts by introducing the notion of a correspondent user interface for services. This is a completely bottom-up approach: the idea is to enrich Web services and resources with UI descriptions (i.e. in UML) to build generic faces to this back-end. Therefore, it takes a completely opposite approach to this paper’s top-down line of attack. ServFace aims to create composite applications from user-created UIs that then are related to an existent enterprise back-end. Taking our approach, UIs are richer, more flexible and closer to end users.

### 8. CONCLUSIONS AND FUTURE TRENDS

In this paper we have described an open component meta-model to show how to use visual and re usable building blocks to easily create and efficiently run distributed systems. It also includes a visual language for composite application development from a user-centered approach. The implementation of any application has been always a complex problem and has required high programming skills. Over the last few years, there has been just a little guideline to help programmers to the development process of these distributed systems, based on the idea of reusability and usage of libraries. The presented approach offers a new open composition model, where end users can exploit their unique expertise in an open innovative creation process. It allows customer without programming skills to build complex composite applications from visual and customizable components.

Future work will concentrate on the development of a formalism that describes both the syntax and semantics of our composition languages. This formalism could be used to verify and validate created instant applications, thus guaranteeing that it arise certain threshold of functionality, reliability, performance, security, clarity, and so on, by an automatic process.

Moreover, we are working on the definition of a taxonomy of forms. Our goal is to find a set of common visual patterns present both in actual UI of Web applications and service front-ends. This set would be a great seed to create a repository of visual UI components that would be exploited to conform any service front-end for any requirements by reusing and connecting them.

### 9. ACKNOWLEDGMENTS

This work is partially supported by the European Commission under the first call of its Seventh Framework Program (FAST STREP Project, grant INF50-ICT-216048) and by the European Social Fund and UPM under their Researcher Training programme.

### 10. REFERENCES
