What characterizes a (software) component?

The definitions and discussions below were contributed via e-mail. They are arranged by date. The following experts, ordered alphabetically, participated in this virtual roundtable during the first quarter of 1998:

Manfred Broy
Technical University Munich
broy@informatik.tu-muenchen.de
http://wwwbroy.informatik.tu-muenchen.de/~broy/

Anton Deimel
SAP AG, Walldorf, Germany
Anton.Deimel@sap-ag.de
http://www.sap-ag.de/

Juergen Henn
IBM, Boeblingen, Germany
jhenn@de.ibm.com

Kai Koskimies
Nokia Research, Helsinki
kai.koskimies@research.nokia.com
http://www.uta.fi/~koskimie/

Frantisek Plasil
Charles University Prague
plasil@nenya.ms.mff.cuni.cz
http://nenya.ms.mff.cuni.cz/~plasil/

Gustav Pomberger
University of Linz
pomberger@swe.uni-linz.ac.at
http://www.swe.uni-linz.ac.at/pomberger/

Wolfgang Pree
University of Constance
pree@acm.org
http://www.swe.uni-linz.ac.at/wolf/

Clemens Szyperski
Queensland University of Technology, Brisbane
c.szyperski@qut.edu.au
http://www.fit.qut.edu.au/~szypersk/

Michael Stal
Siemens AG, Munich
Michael.Stal@mchp.siemens.de

Anton Deimel:

A component:
1. Represents one or more logical or organization-related
   processes or tasks (for example, purchasing, sales or
   management of master data)
2. Is more coarse-grained than single classes; in other words,
   a component usually consists of several logically coherent classes
3. Is unique from other components because a class can be
   assigned only once to a component.
4. May consist of other components.
5. Uses precisely-defined interfaces to communicate with other
   components
6. Is independent of the release (components can be upgraded
   and distributed) and can be delivered separately
7. Frameworks form the underlying technology for componts
8. May be a client and server for other components.

Points 1 - 4 come from an article by Dr. Wolfgang Hesse and
Prof. Dr. Friedrich Weltz, entitled "Project Management for
Evolutionary Software Development".

Wolfgang Pree, Gustav Pomberger:

Our definition of components is derived from taking a look at the deficiencies
of the object-oriented paradigm:

* Classes/objects implemented in one programming language cannot
interoperate
with those implemented in other languages.

* Composition of objects is typically done on the language level. Black-box
composition support is missing, that is, visual/interactive tools that allow
the plugging together of objects.

Characteristics of components:

* A component is simply a data capsule. Thus information hiding becomes the core construction principle underlying components.

* A component can be implemented in (almost) any language, not only in any module-oriented and object-oriented languages but even in conventional languages.

* As a consequence, standards how to describe componets have been established.
The component (module) interface is described either
  - textually by means of an interface description language (IDL)
  - visually/interactively by appropriate tools.

* Framework architectures form the enabling technology of plug&play software, where most adaptations can be achieved by exchanging components. Visual, interactive composition tools are ideally built on top of domain-specific frameworks.

Single component reuse is less attractive. It means that programmers build the overall software system architecture on their own. They have to locate the appropriate components in a lego kit library and define their interactions.

Kai Koskimies:

It it probably very difficult to cover all aspects that are associated with components in
different contexts. There are lists of component features in some textbooks trying to
do this, but I suspect that they are rather long because the authors themselves are a bit
unsure of the essence of a component. And so am I. Anyway, I would suggest something like
the following:

A component is a system-independent binary entity which implements one or more interfaces.

An interface is a collection of signatures of services belonging logically together.

A point here is that since a component implements interfaces, it can be plugged in
to any context requiring one of those interfaces. System-independency means

interoperability with respect to languages, platforms, applications, tools etc. The
fact that an interface consists of logically related services implies that a component
has a "meaning", so to say, i.e. an identifiable role in the system it is plugged into.
As far as I can see, this definition does not contradict with Wolfgang's characterizations.

BTW, I noted that Oscar Nierstrasz has used the definition: "static abstraction with plugs"
That is perhaps too terse for my taste.

Clemens Szyperski:
Here is a compact definiton that we developed in the first Workshop on Component-Oriented Programming (WCOP'96) at ECOOP'96 in Linz:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. [WCOP'96 Summary in ECOOP'96 Workshop Reader, dpunkt Verlag, 1997. ISBN 3-920993-67-5]

The entire one day workshop mostly concentrated on producing this definition which since then has survived intact.

Michael Stal:

to complete the definitions parade I want to just add my own definition:

A component is a binary unit that exports and imports functionality using a standardized interface mechanism. The underlying component infrastructure supports composition of components by providing mechanisms for introspection, event-handling, persistence, dynamic linking and layout-management.
In general, application frameworks are required for building components as well as for composing them.

What about this definition that incorporates what componentware technologies such as Opendoc, ActiveX Controls and JavaBeans provide.

Frantisek Plasil:
in addition to CORBA and dynamic component updating (mainly in Java),
my research interests include Architecture Description Languages(ADLs).
Below pls find comments on the component concept from this perspective.

>From szypersk@fit.qut.edu.au Sat Jan 31 05:01 MET 1998
> Here is a compact definition that we developed in the first Workshop on
> Component-Oriented Programming (WCOP'96) at ECOOP'96 in Linz:
>
>   A software component is a unit of composition with contractually
>   specified interfaces and explicit context dependencies only. A
>   software component can be deployed independently and is subject to
>   composition by third parties.

> A crucial point that I missed in Kai's two-liner: components that are at
> all useful CANNOT be entirely context-independent. However, they have to
> come with a clear specification of what it is that they depend on:
>
>   (i) required interfaces, that is, services they need but don't provide
>   themselves,

- Sure, in ADLs
  a component specification typically includes the "provides" and
  "requires" clauses (interface(s)).

- components are interconnected: (provides - requires connections)

  to cover different paradigms of interconnection under one roof, ADLs
  come up with the "connector" abstraction. Different types of
  connectors may coexist in one architecture (RPC, events, pipe, ORB,
  ...). Connectors also solve the interface adaptation problems;
  informally, there is a number of adaptors (method granularity) in a
  connector (interface granularity).

- components can be nested
> From anton.deimel@sap-ag.de Tue Feb 3
> 4. May consist of other components.

  this helps with structuring of the architecture being designed, and,
  of course, encapsulates the functionality of nested components.
  Overall, nesting influences granularity. Thus:

>From pree@acm.org Fri Jan 30 10:27 MET 1998

> Single component reuse is less attractive. It means that programmers build the
> overall software system architecture on their own. They have to locate the
> appropriate components in a lego kit library and define their interactions.

works fine in the case nesting is not allowed and there is a separate
abstraction to capture component composition ("framework" usually). As
an aside, Jave Beans cannot be nested; however, the new version of Java
Beans, Glasgow, will allow for nesting of beans.

- framework issues

>From pree@acm.org Fri Jan 30
> * Composition of objects is typically done on the language level. Black-box
> composition support is missing, that is, visual/interactive tools that allow
> the plugging together of objects.

> * Framework architectures form the enabling technology of plug&play software,
> where most adaptations can be achieved by exchanging components.

>From this I understand that W has the black-box frameworks
in mind (also:"Black-box...modifications are accomplished by
composition not by programming "(W's SIGS book, p.22)). Thus a composed
component CC created by nesting (and composing) of other components can
be viewed as a (black-box) framework M, if the internal architecture of
CC is revealed and there is a way/tool for modification of M alias CC
by (re)composition. This can include exchanging some of the CC
subcomponents. This, of course, contradicts the idea of "binary"
character of the CC component. As an aside, the CC "plug in" operation
can be view as instantiation of M.

We use this idea in our SOFA/DCUP architecture; the modification
operation is called update and can be even done at runtime.

- binary character
>From kai.koskimies@research.nokia.com Fri Jan 30
> A component is a system-independent binary entity ...

> From szypersk@fit.qut.edu.au Sat Jan 31
> ... . As an aside:
> "binary" is not to be taken too literally - there is nothing wrong with
> source delivery subject to on-the-fly compilation. The point is that
> source-level "reuse" is not intended.

In my view, an agreement for revealing the internals of a component may be granted by the provision policy (part of the component provider - customer contract). In a component nesting scenario, some of the components can be "private" some "revealed" = open to modification by the customer. (Analogy of source code /binary SW license). Than, the revealed components can play the role of framework(s))

- the environment (platform) vers. system independence

>From kai.koskimies@research.nokia.com Fri Jan 30
> A component is a system-independent binary entity which implements one
> or more interfaces.
> ...to any context requiring one of those interfaces. System-independency means
> interoperability with respect to languages, platforms, applications, tools
> etc.

>From michael.stal@mchp.siemens.de Mon Feb 2
> .. using a standardized interface mechanism. The underlying component
> infrastructure supports composition of components by providing mechanisms
> for introspection, event-handling, persistence, dynamic linking and
> layout-management.

I am not sure if we do not ask too much by this requirement of general system-independency. In my view, of course there is an underlying infrastructure which provides services as Michael points out. Different platforms may provide different services, e.g. the event models in CORBA and Java differ.

A nice abstraction that clearly separates the problem of reuse of objects (components) from portability of services is the meta-level architecture and/or meta-object protocol (e.g.Uni of Tokyo: The Apertos OS group, TCDublin: V.Cahill, Xerox PARC: G.Kiczales). The other issue related to system-independence/portability is that a service might be orthogonal to a component's code in one implementation and another implementation may need a support from the component. A typical example is persistency: if you use pJava, persistence is achieved automatically, if you port the component to CORBA you might end up with providing method for accessing the state of the objects involved in the component. Using the meta-level abstraction avoids the problem (the burden of porting is limited to the meta-level implementation only).

Finally,

> From szypersk@fit.qut.edu.au Sat Jan 31 05:04:59 1998
>
> Now that you are all on your way ordering my book :) - here's a final
> incentive: I also list quite a number of "other" component definitions,
> including Oscar's.

I did (www.books.com) 14 days ago . However, they say the book should
be on their shelf not earlier than Feb.4., so it will take 7 - 10 days
to get it. In the meantime, Clemens, if there was a chance to get from
you a ps file of the chapters 4,5,11,20,21 it would be a speed up with
respect to our discussion. Naturally, if there are any legal and/or
technical problems, I understand.

To summarize, my component characteristics include:

1. requires/provides interfaces (plus contract description)
2. component nesting allowed
3. revealing of a component's internals may be granted by provision policy
   (providing the component as a framework)
4. may be platform specific (platform independence is welcome)

Manfred Broy:
Let me be very provocative. All what I read does not define
the notion of a component. It says a bit about properties
one might expect from a component. But this is not enough.

I would like to see a general definition of the notion
of a component which is i n d e p e n d e n t of all
the technical stuff (saying what is a component in
Java or CORBA or whatever).

Have a look at my article to see what I am aiming at:

http://www4.informatik.tu-
muenchen.de/papers/Broy_CUC1996_klein_1996_Publication.html

P.S: Below are my comments on the early messages:

| Date: Fri, 30 Jan 1998 09:47:43 +0200
| From: Wolfgang Pree

||
| Characteristics of components:
|
| * A component is simply a data capsule. Thus information hiding becomes the
| core construction principle underlying components.
|
| * A component can be implemented in (almost) any language, not only in any
| module-oriented and object-oriented languages but even in conventional languages
| .

This all does not define a component!

|
| * As a consequence, standards how to describe componets have been established.
| The component (module) interface is described either
| - textually by means of an interface description language (IDL)
| - visually/interactively by appropriate tools.
|

Description must not only mean the syntactic interface!

 | From: kai.koskimies@research.nokia.com (Koskimies Kai NRC/Hki)
| Date: Fri, 30 Jan 1998 19:23:31 +0200
| Subject: RE: component definition
|
|
| It it probably very difficult to cover all aspects that are associated with
| components in
| different contexts. There are lists of component features in some textbooks
| trying to
| do this, but I suspect that they are rather long because the authors
| themselves are a bit
| unsure of the essence of a component. And so am I. Anyway, I would suggest
| something like
| the following:
|
| A component is a system-independent binary entity which implements one
| or more interfaces.
|

What does "binary" mean here?
What is the difference between one and two interfaces?
If interfaces are collections, we should be able to join

two interfaces into one.

| An interface is a collection of signatures of services
| belonging logically together.

What is a "signatures of services"?
 | From: Clemens Szyperski <szypersk@fit.qut.edu.au>
||
| A software component is a unit of composition with contractually
| specified interfaces and explicit context dependencies only. A
| software component can be deployed independently and is subject to
| composition by third parties. [WCOP'96 Summary in ECOOP'96
| Workshop Reader, dpunkt Verlag, 1997. ISBN 3-920993-67-5]
|

How is the interface specified technically?

| From: michael.stal@mchp.siemens.de
| Date: Mon, 2 Feb 1998 08:30:10 +0100 (MET)

|
| A component is a binary unit that exports and imports functionality
| using a standardized interface mechanism. The underlying component
| infrastructure supports composition of components by providing mechanisms
| for introspection, event-handling, persistence, dynamic linking and
| layout-management.
| In general, application frameworks are required for building components
| as well as for composing them.
|

This definition uses a lot of terms that have to be explained.
For my taste it is a bit too dependent on a special technology.

>From anton.deimel@sap-ag.de Tue Feb 3 17:5
| Date: Tue, 3 Feb 1998 17:44:17 +0100 (MET)

| Definition of Component:
|
| A component:
| 1. Represents one or more logical or organization-related
| processes or tasks (for example, purchasing, sales or
| management of master data)
| 2. Is more coarse-grained than single classes; in other words,
| a component usually consists of several logically coherent classes
| 3. Is unique from other components because a class can be

| assigned only once to a component.
| 4. May consist of other components.
| 5. Uses precisely-defined interfaces to communicate with other
| components
| 6. Is independent of the release (components can be upgraded
| and distributed) and can be delivered separately
| 7. Frameworks form the underlying technology for componts
| 8. May be a client and server for other components.
|

This again lists a number of properties without saying what a
component is.

| From: kai.koskimies@research.nokia.com (Koskimies Kai NRC/Hki)
| Date: Tue, 03 Feb 1998 19:51:13 +0200
| I liked the "derived" definition of Clemens more than the workshop
| definition,
| maybe group work does not always produce the optimal result... Michael's
| definition sounded good, but perhaps the list of supporting mechanisms
| need not be so exclusive?
|
| Clemens, I hope you do not mind if I try to attack the workshop definition:
|
| > A software component is a unit of composition
| I think a component is a unit of composition by the definition of the word
| > with contractually specified interfaces
| "contractually specified" needs explanation
| Is interface a part of a component? This seems to imply so, but I would like
| to see interfaces as separate entities
| > and explicit context dependencies only.
| This part is good
| > A software component can be deployed independently
| Independently of what? It needs anyway the explicitly specified context
| > and is subject to composition
| Again, isn't it trivially true that a component is subject to composition
| > by third parties.
| What is the first and second party? Why the identity of the component user
| is relevant?

I agree very much with that attack.

| From: plasil@nenya.ms.mff.cuni.cz
| Date: Thu, 5 Feb 1998 15:09:52 +0100
| I am not sure if we do not ask too much by this requirement of general
| system-independency.

I am very intersted in an independent notion of a component.

| To summarize, my component characteristics include:
|
| 1. requires/provides interfaces (plus contract description)
| 2. component nesting allowed
| 3. revealing of a component's internals may be granted by provision policy
| (providing the component as a framework)
| 4. may be platform specific (platform independence is welcome)
|

Again this does not solve the problem, what a component is.

Michael Stal:

> | A component is a binary unit that exports and imports functionality
> | using a standardized interface mechanism. The underlying component
> | infrastructure supports composition of components by providing mechanisms
> | for introspection, event-handling, persistence, dynamic linking and
> | layout-management.
> | In general, application frameworks are required for building components
> | as well as for composing them.
> |
>
> This definition uses a lot of terms that have to be explained.
> For my taste it is a bit too dependent on a special technology.
>

This definition was established summarizing several years of experience
with Componentware issues. I do not see why it is too dependent on a
specific technology. However, I agree with your point that the other terms
need some explanations (I assumed that these terms are general knowledge).

It might help us to get some hints where from your point of view
technological dependencies are interwoven with the definition.

Another problem I like to emphasize is the point that we are trying to define
something that is still undefined and unclear and where no agreement
can be made.
Everyone of us does have another perspective on the notion of components.
That is pretty the same like defining other terms such as "object" or
"god". Thus, in some sense every definition suggested so far is correct.

But, what is the consequence of this problem? We can either specify a

very detailed but also very narrowed component definition. Or we
could otherwise give a very high-level and abstract component definition
that fits everything such as

"A component is a binary software module that exports and imports
functionality using a standardized interface mechanism. Point."

I am not biased to any of these alternatives, but I think we should
better have a detailed and more restrictive definition, because otherwise
our discussions would be not very fruitful.

Kai Koskimies:
I am not sure if I have got all the messages, but as far as I see our task
has not
made very good progress. Perhaps we should decide some procedure now
before
going
on with the discussion.

1) Definition of a component

I am not any more sure that we understand this term in the same way. Let me
make a simple
test: could you all tick those items below that you consider as a component:

a) a subroutine in a Fortran mathematical library
b) a class in Java
c) an Ada library package
e) an Oberon module
g) a COM object
h) a Java Bean
i) any application that can be used by another application through a
specified interface

If we have problems in achieving a common definition, one possibility would
be to
define different levels of "component-orientation", in the same way as Peter
Wegner
did in the article defining "object-orientedness" (object-based,
class-based,
object-oriented). We all seem to agree that a component is associated with
an
interface, but properties like linking-time (static, dynamic) and
platform-independency

are less clear. If we can find a natural hierarchy of components, that might be
a real contribution and clarify the area in general.

Manfred Broy:
Here are my reactions to recent e-mails

Manfred

| From: kai.koskimies@research.nokia.com (Koskimies Kai NRC/Hki)
| Date: Mon, 23 Feb 1998 17:08:47 +0200
|
| 1) Definition of a component
|
| I am not any more sure that we understand this term in the same way. Let me
| make a simple
| test: could you all tick those items below that you consider as a component:
|
| a) a subroutine in a Fortran mathematical library
| b) a class in Java
| c) an Ada library package
| e) an Oberon module
| g) a COM object
| h) a Java Bean
| i) any application that can be used by another application through a
| specified interface

I think all these can be seen as examples of components. However,
I would be interested in a more general notion of a component - language
independent (at least as a concept), freely combinable and executable
on distributed plattforms.

| From: michael.stal@mchp.siemens.de
| Date: Mon, 23 Feb 1998 10:42:06 +0100 (MET)
| This definition was established summarizing several years of experience
| with Componentware issues. I do not see why it is too dependent on a
| specific technology.

I think a lot of the discussion is coined by specific programming
languages or specific middleware. Is that what we want?
Or do we aim at a more generic universal notion of a component?

| Everyone of us does have another perspective on the notion of components.
| That is pretty the same like defining other terms such as "object" or
| "god". Thus, in some sense every definition suggested so far is correct.

I do not think that the word "correct" is correct. Our definition should
not be correct but capture the engineering intension.

| But, what is the consequence of this problem? We can either specify a
| very detailed but also very narrowed component definition. Or we
| could otherwise give a very high-level and abstract component definition
| that fits everything such as
|
| "A component is a binary software module that exports and imports
| functionality using a standardized interface mechanism. Point."

That does not help very much, because it is too general and too
low level (why binary?).

| I am not biased to any of these alternatives, but I think we should
| better have a detailed and more restrictive definition, because otherwise
| our discussions would be not very fruitful.

I agree!

Michael Stal:
thanks a lot for your comments.

To me it seems that you already have a component definition in mind.
Now, it is time to see your definition of component, because that may help
us with our own efforts.

What I'd additionally like to see is how other technological areas define
components, e.g., ICs in the the electronics domain. Are there any
components in sciences such as mathematics or physics? Maybe, some of
you
could get some input.

Wolfgang Pree:

Manfred Broy wrote:
> Description must not only mean the syntactic interface!

Michael Stal wrote: (replying to comments of Manfred Broy regarding his definition)
> This definition was established summarizing several years of experience
> with Componentware issues. I do not see why it is too dependent on a...

As the last threads of discussion corroborate, we can clearly discern between the syntactic and the semantic aspects of components. (Probably, this could form an initial classification scheme as proposed by Kai Koskimies.) Currently, it seems that only syntactic issues are addressed by component standards. In my opinion, this represents an important first step.
On the other hand, I fully agree that this is not sufficient and that semantic aspects are even more important after these nasty basic problems have been solved. Unfortunately, almost no practically relevant concepts and tools are around so far to tackle the problems related to semantic issues.

Michael Stal:
I've tried a new definition of the term component. Here is a definition which is not technology-biased as my former definition.

Component Definition

A component denotes a self-contained enity (a.k.a. black box) that exports functionality to its environment and may also import functionality from its environment using well-defined and open interfaces. In this context, an interface defines the systax and semantics of the functionality it comprises (i.e., it defines a contract between the environment and the component).
Components may support their integration into the surrounding environment by providing mechanisms such as introspection or configuration functionality.

Jürgen Henn:
I watched most of the discussions and found it quite interesting to see the different definitions, compared to what I had in mind and how we practice components. To be honest, I never before thought of a component being something
like a Fortran subroutine, but I can very well understand the rational behind this. It definitely has to do with the context of your thinking and the environment you are working in.

In such a case, object people (and I consider myself as being one of them)
start to build abstractions and specialisations. So why not defining a
hierarchy of 'component types' and try to attach the different definitions to
the different levels?
My usage of component is more related to object technology and business
applications. An 'object component' could be a specialisation of a general
component, and an 'application component' again is a specialisation of an
'object component'. Object components represent a collection of strongly
related classes which form a functional, encapsulated unit to the outside world
using well defined interfaces and behaviour. They are elements for
development,
versioning, maintenance, etc. Application components extend this definition by
integrating application services like persistency, transaction handling,
security, presentation, etc. For example, an application component represents
the granularity to define access rights or to define transactional context.
These kind of components are primarily used because single classes do not
provide the right granularity to structure applications. They require a
component framework (which also consists of cooperating classes) as
component
infrastructure which defines the overall component structure and offers
common
services.

Wolfgang Pree:

Juergen Henn wrote:
> My usage of component is more related to object technology and business
> applications. An 'object component' could be a specialisation of a general
> component, and an 'application component' again is a specialisation of an
> 'object component'. ...
I don't see the difference between object components and application
components. The distinctive feature of application components that you
mention
is appl. services. Then you list persistency, transaction handling, etc. as
sample application services. I view these aspects as domain independent and
not application specific.

Furthermore, your categorization does not answer the question what
components
are. You seem to use components as synonymous term for objects/classes.

Dear colleagues,

we all are very busy and the discussion did not really
progress as we might have wished.

Therefore I add some material taken from an article
to make my position clear.

XXX Postscript file, attached separately

 Michael Stal:
Hello Prof. Brof,
Hello Colleagues,

thanks a lot for your document.

>From my viewpoint the definition you provided reveals one
possible definition for a specific kind of components. Interestingly,
there are some commonalities between the definition I provided some
weeks ago and yours. Maybe, there is a way to find a
common abstraction we all could agree on. I think, everyone of us
made some useful suggestions on specific viewpoints of components.
We should now try to find a common denominator.

What could help us in this context is a mapping of your component
definition to existing component technologies such as JavaBeans or
ActiveX Controls.

Manfred Broy:

I would be very interested in more specific comments on
my proposal of a component model. Taking into
account Michael's comments I claim that
Java Beans or Active X components can also be represented
that way (forgetting about inheritance, which basically
is code reuse and therefore cannot be dealt with in
an sufficiently abstract model).

Wolfgang Pree:

Manfred Broy wrote:
> I would be very interested in more specific comments on
> my proposal of a component model. Taking into
> account Michael's comments I claim that
> Java Beans or Active X components can also be represented

> that way (forgetting about inheritance, which basically
> is code reuse and therefore cannot be dealt with in
> an sufficiently abstract model).
I agree that Java Beans and ActiveX components can be
represented this way. My concern is that the definition
seems to be too general/abstract. As you mention, inheritance
is neglected, which is not a problem per se. The point is that
it becomes impossible to tell what distinguishes a component
from an object. Or don't you see any differences?

I also don't see the benefits of describing the semantic
aspects of a component by merely considering its input/output
behavior. Could, for example, different components test each
other based on this semantic information?

Manfred Broy:

Dear Wolfgang,

I am very grateful for your responds because it mentions a number of
interesting aspects of components that should be discussed. Of course, you
are right that my definition of a component is rather general. This way be seen
as a disadvantage but on the other hand
nevertheless it has enough semantic power to include all kinds of
components notion. As soon as you go down to a specific programming
language or a specific component paradigm - of course - the notion has to
be specialised in particular if we are interested in more
practical aspects of component description or reasoning about components.
But - if componentware is about combining all kinds of components
from different programming paradigms a very general comprehensive
notion is badly needed.

Another aspect is inheritance. Inheritance as it is found in most object
oriented programming languages is mainly code-inheritance. Therefore
it breaks all rules and principles of information hiding because
inheritance of code means in most cases that you have to be aware of the
implementation details of the classes and objects involved.

On the other hand there are ideas around to understand inheritance in a
more modular way than code inheritance as property inheritance.
In this case inheritance can be
perfectly incorporated into my suggested model of a component.

Your question whether a object is a component or what a component

distinguishes from an object is an interesting one. This again has to do
with the question: what is a component and how general should this notion
be?
>From my point of view an object can
be perfectly seen as a - in general - rather small component. In most cases
it would be interested in larger components. But I see no reason why such
larger components should not also be understood as objects. Especially if
we have a bit more general notion of an object as been something which may
be
also composed of subobjects then a component in an object oriented
framework
perfectly be a component in vice versa.

Finally to your remark: what are the benefit of describing the semantic
aspects of a component by merely considering its input/output behaviour? I
think this is all what modularity is about. If you are only interested as a user
in a component but not to understand its implementation all you have to know
is
its input/output-behaviour. I see no reason why we should not describe this
input/output-behaviour also in a way such that different components can
make use that information when trying to configure a system.