Composition and Refinement for Partial Object Specifications

Einar Broch Johnsen and Olaf Owe
Department of informatics, University of Oslo
PO Box 1080 Blindern, N-0316 Oslo, Norway
{einarj, olaf}@ifi.uio.no

Abstract

For the specification and development of large, distributed, and object-oriented systems, it is often advocated that individual components should be developed in an aspect-wise manner, where separate descriptions depict various roles or viewpoints of the objects considered. The introduction of such partial specifications requires extra care when reasoning about systems as several specifications of an object may coexist and lead to overlapping information.

In this paper, we consider a compositional approach to system development by means of partial specifications of objects. The approach supports stepwise refinement, which enables global reasoning by local refinement steps in an aspect-oriented specification style. For this purpose, a refinement relation is proposed which is suitable for multiple inheritance of behavior and component upgrade.

1 Introduction

For the development of complex object-oriented distributed systems, a formal specification language should support both compositional and stepwise reasoning. In a compositional formalism, the specification of a system can be obtained from the specifications of its constituent components, without knowledge of the interior construction of those components [27]. Refinement relations ensure the correctness of specification development steps. The two notions achieve opposite purposes in the sense that refinement represents a top-down approach to system development and composition is bottom-up. Whereas refinement adds detail to specifications in order to make them more precise, composition encapsulates internal activity and results in a more abstract view of the system under development. An abstract specification may be refined by a composition of several (more low level) specifications. With a compositional refinement relation, global reasoning can be performed by local refinement steps. Compositional refinement properties have been studied in different formalisms, see e.g. [5, 9, 18].

A new feature in this paper is that compositional refinement properties are studied in a setting where several specifications of the same object can be given, focusing on different aspects or roles associated with that object. A multiple viewpoint approach to system development is advocated for open distributed systems [12] and aspect-oriented programming [15], and to some extent supported through e.g. Java interfaces, Corba IDL [19] and UML interfaces and roles [8], without formalization. To illustrate how compositionality and refinement relate to each other in such a setting, we use a simple object-oriented specification notation designed to capture exactly these issues.

We consider *open* distributed systems where objects run in parallel, communicate by remote method calls, and exchange object identities. For such systems, we do not have local control of the components. Instead, the behavior of a component is locally determined by its interaction with the environment [2]. This *observable behavior* of an object or component gives us an abstract view of its state, hiding encapsulated implementation details. The life of an object up to a point in time is recorded in its trace, i.e. the finite sequence of observable communication events reflecting remote method calls both to and from the current object. Trace descriptions of process and data flow networks as well as modules are known from the literature [9, 11, 14, 17, 22]. The set of traces that reflect possible runs of an object up to points in time, give us an observational description of that object and is prefix-closed. Prefix closed trace sets capture safety properties [3]. For simplicity reasons, liveness is not considered in this paper.

A safety specification includes an alphabet of communication events and a prefix closed set of traces over this alphabet. When an object is specified at a given level of abstraction, some lower level details concerning its behavior are ignored, in order to focus the specification on a relevant aspect of the behavior of the object. Such details may be a subset of the methods offered by the object or a subset of its communication with other objects in its *environment*. To capture the dynamic character of open systems, the number of objects in the environment is (potentially) infinite.

Proc. 16th Intl. Parallel & Distributed Processing Symposium (IPDPS 2002), Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA 2002), IEEE Computer Society Press, April 2002.

Multiple viewpoints of an object can result in several specifications that consider different sets of communication events. We propose a refinement relation that allows alphabet expansion, i.e. the addition of new events through refinement. Thus, various (partial) specifications of an object may have a common refinement, although their alphabets differ. This refinement relation is introduced in the OUN specification language [10, 20] and has some resemblance to extensional subtyping [16]. In this paper, we generalize this refinement relation to the setting of components and allow the introduction of new objects in a refinement step. We find that the relation gives us reasoning control over functionality upgrades for components, and reflects the intention of (multiple) inheritance in object-oriented programming languages.

A component encapsulates an arbitrary number of objects. Due to this encapsulation, internal communication is hidden from an external observer. In order to reason about viewpoint specifications of components, it is necessary to consider both observable activity and internal events. We investigate the interaction between compositionality and refinement in this setting. Partial descriptions and object identities add difficulty to the problem of obtaining compositionality, as the same object identifiers may appear in several specifications.

The paper is organized as follows. The next section introduces the trace-based specification notation employed in the paper. Section 3 introduces a refinement relation which we find suitable for viewpoint specifications. In Section 4, we show how encapsulation works for the composition of specifications of single objects and, in Section 5, compositional refinement is shown for this case. Then, in Sections 6 and 7, the formalism is generalized to components and compositional refinement is shown for component specifications when some conditions are satisfied. Finally, in Sections 8 and 9, we motivate our refinement relation by examples and discuss the proposed solution in relation to other work. A full version of this paper, with proofs and additional examples, is available as a research report [13].

2 A Formalism for Partial Object Specifications

Objects are not static parts of a system, but evolve through interaction with the environment. One may think of the current "state" of an object as the result of its past interactions with the objects of the environment by way of remote method calls. Internal activity in an object is reflected by nondeterminism in its observable activity. Objects are modeled by the semantic concept of finite communication traces, known from for example CSP [11], but the communication events are extended with information about the identities of the sender and receiver of the remote calls.

A communication trace is a finite sequence over an alphabet of communication events. Let Mtd and Obj denote types for methods and object identities, respectively. Say that a method m is provided by an object o_1 and that this method of o_1 is called by another object o_2 (ignoring method parameters). This call is represented by a *communication event*, which is a triple $\langle o_2, o_1, m\rangle$, where $o_1, o_2 \in Obj$ and $m \in Mtd$. When an object calls methods in itself, this activity is understood as internal and is as such not reflected in the alphabet nor in the traces of the object. In an observable event $\langle o_2, o_1, m\rangle$ we can assume that $o_1 \neq o_2$. We denote by α^o the set of possible observable communication events of an object o, defined as follows:

$$\alpha^o = \{\langle o_2, o_1, m\rangle \mid m \in Mtd \wedge o_1, o_2 \in Obj \\ \wedge o \in \{o_1, o_2\} \wedge o_1 \neq o_2\}.$$

Denote by $\mathcal{T}^o$ the semantically defined set of traces over this alphabet which describe all possible executions of the object o, so $\mathcal{T}^o \subseteq \text{Seq}[\alpha^o]$, where $\text{Seq}[S]$ denotes the set of finite sequences over a set S.

A specification of an object is a *partial* description of that object. Thus, we allow several specifications of the same object. To each specification Γ of an object o, we associate an alphabet α that consists of those communication events of o that are considered by Γ, so $\alpha \subseteq \alpha^o$, and a trace set over α. We now define the notion of a specification of a set of objects $\mathcal{O}$:

Definition 1 *A specification Γ is a triple $\langle \mathcal{O}, \alpha, \mathcal{T}\rangle$ where $\mathcal{O}$ is a finite set of object identities, $\mathcal{O} \subseteq Obj$, α is an infinite set of events such that*

$$\alpha \subseteq \{\langle o_1, o_2, m\rangle \in \bigcup_{o\in\mathcal{O}} \alpha^o \mid \neg(o_1 \in \mathcal{O} \wedge o_2 \in \mathcal{O})\},$$

and $\mathcal{T}$ is a prefix closed subset of $\text{Seq}[\alpha]$.

We will call $\mathcal{O}$ the object set of the specification Γ, α the alphabet of Γ, and $\mathcal{T}$ the trace set of Γ. In shorthand, these will be referred to as $\mathcal{O}(\Gamma)$, $\alpha(\Gamma)$, and $\mathcal{T}(\Gamma)$, respectively.

As the trace sets are prefix closed, it follows that we specify safety properties in the sense of [3]. In the examples, the trace sets will be given by predicates. A trace set defined by a predicate P on traces $h : \text{Seq}[\alpha]$ is the largest prefix closed subset of $\{h : \text{Seq}[\alpha] \mid P(h)\}$.

Although the alphabets of specifications are statically defined, the communication of object identities can still be modeled by a particular object's ability to communicate with another object at a given point in history. This is reflected in the trace set.

Every specification has a *communication environment*, which is the set of objects involved in communication with objects of the specification. The communication environment can be derived from the specification; for a specification Γ, it is the set $\{o : Obj \mid o \notin \mathcal{O}(\Gamma) \wedge (\langle o, o', m\rangle \in$

$\alpha(\Gamma) \vee \langle o', o, m\rangle \in \alpha(\Gamma))\}$. For *open* systems, we do not control the environment and new objects can appear in the environment at any time. The communication environment (and therefore the alphabet) of a specification is infinite.

If the object set of a specification Γ is singleton, say $\mathcal{O}(\Gamma) = \{o\}$, we call Γ an *interface* specification (of the object o). In the literature, the soundness of a specification Γ of an object o is given by $\mathcal{T}^o \subseteq \mathcal{T}(\Gamma)$ (see e.g. [4]). In our case, we allow partial specifications of objects, so $\alpha(\Gamma) \subseteq \alpha^o$. Consequently, the traces must be restricted to the alphabet considered in the specification, i.e. Γ is a sound specification of o if $\forall h \in \text{Seq}[\alpha^o] : h \in \mathcal{T}^o \Rightarrow h/\alpha(\Gamma) \in \mathcal{T}(\Gamma)$. Soundness for (multi-object) component specifications is discussed in Section 6.

Trace Notation. Filtering functions on traces are needed. Let h be a trace and S a set of events. Then h/S denotes the trace obtained by deleting all events in the trace h that are not elements of S and $h \setminus S$ denotes the trace obtained by deleting all events that are elements of S. By extension, h/o will represent the restriction of a trace h to the set of events involving an object o.

Example 1 Consider an (interface) specification Write of an object o controlling write access to some shared data, following [10]. Access is restricted so that only one object in the environment may perform write operations at the time. Let $Objects$ be a subtype of Obj not containing o and let $Data$ be a set of data. The specification only considers one object, so $\mathcal{O}(\text{Write}) = \{o\}$. There are methods for writing, opening, and closing write access, which we name W, OW, and CW, respectively. The write method W has a parameter ranging over $Data$. The alphabet[1] of Write is then

$$\alpha(\text{Write}) \triangleq \{\langle x, o, OW\rangle, \langle x, o, CW\rangle \mid x \in Objects\} \cup \{\langle x, o, W(d)\rangle \mid x \in Objects \wedge d \in Data\}.$$

Controlled write access is obtained by restricting the possible traces of Write. Define the trace set as follows:

$$\mathcal{T}(Write) \triangleq \{h : \text{Seq}[\alpha(\text{Write})] \mid h \textbf{ prs } [[\langle x, o, OW\rangle \langle x, o, W\rangle^* \langle x, o, CW\rangle] \bullet x \in Objects]^*\}.$$

Here, the predicate h **prs** R denotes that the trace h is a prefix of the regular expression R. We denote by R^* the repetition of a regular expression R and by $R_1\ R_2$ the sequential composition of expressions R_1 and R_2. Expressions may be grouped by brackets $[\ldots]$. In addition, $\bullet$ is a binding operator. In this example, x is bound for each traversal of the loop. In the trace set, the binding operator on calling objects ensures sequential write access. A caller may perform multiple write operations once it has access. Note that a set defined by a predicate h **prs** R is always prefix closed.

[1] A call to $W(d)$ can be modeled by two events where only the first contains the value which is written. This lets us capture asynchrony [20].

3 Refinement

As a specification evolves, details hitherto ignored are added in order to incorporate postponed design choices in the specification. We will refer to the process of adding details concerning behavior to a specification as refinement. Other details such as refinement of method parameters may be handled by abstraction functions, which we do not consider here. When the refinement relation is formalized, criteria for a "correct" development process is obtained.

A formal refinement relation usually states that a specification Γ' refines another specification Γ if Γ' implies Γ, i.e. $\mathcal{T}(\Gamma') \subseteq \mathcal{T}(\Gamma)$ for trace-based specification languages. This concept of refinement is found in e.g. Action Systems [5], CSP [23], FOCUS [9], and TLA [1]. The relation expresses that the concrete specification Γ' is more precise than the abstract specification Γ. An essential principle of this kind of refinement is that a trace should have the same possible extensions in both specifications, i.e. a call to an object that was considered in the abstract specification should generate a response from the object in the concrete specification that was already possible in the abstract one. With viewpoints, this principle is not satisfactory.

It is natural to perceive viewpoint refinement of interface specifications as a step towards considering the full behavior of the object. A specification inherits the alphabet and behavior of its ancestor through refinement, but we will also allow the addition of new methods and new requirements (additional behavioral restrictions). In a refinement step, we then need to expand the alphabet of a specification, but still restrict its possible behavior (when the new method calls are ignored). This idea resembles the notion of extension in behavioral subtyping proposed by [16], when new methods are not interpreted at the abstract level. For components, we allow inclusion of new object identifiers in addition to new methods. We now define refinement.

Definition 2 *Let Γ and Γ' be specifications. Γ' refines Γ, denoted $\Gamma' \sqsubseteq \Gamma$, if 1) $\mathcal{O}(\Gamma) \subseteq \mathcal{O}(\Gamma')$, 2) $\alpha(\Gamma) \subseteq \alpha(\Gamma')$, and 3) $\forall h \in \mathcal{T}(\Gamma') : h/\alpha(\Gamma) \in \mathcal{T}(\Gamma)$.*

Two specifications of the same object identifier can be refined into a single specification, although the specifications may have disjoint alphabets. In this way, we obtain multiple inheritance for specifications. For partial specifications, this is particularly attractive. The approach is well suited for open distributed systems, where components are replaced or extended on the fly [20].

The refinement relation given here is a partial order. For refinement of specifications without alphabet expansion or object addition, this refinement relation coincides with the traditional subset-relation and solely restricts the allowed behavior of the old specification, making the new specification more deterministic. If new objects are included in a

refinement Γ' of Γ, it follows that these objects cannot be in the communication environment of Γ. The introduction of fresh object identifiers, corresponding to the **new** command of object-oriented languages, is always allowed.

We illustrate this definition of refinement by two examples.

Example 2 Consider interfaces Read and Read2 for concurrent read access to the shared data resource controlled by the object o of Example 1. Read has one method R which offers $Data$ values to objects in the environment and a trace set $\mathcal{T}(Read) = \{h{:}\,\mathrm{Seq}[\alpha(\mathrm{Read})]\}$. In addition to R, Read2 uses open_read and close_read operations OR and CR and has an alphabet similar to Write. Define the trace set of Read2 by the predicate

$$\forall x \in Object : h/x \textbf{ prs } [\langle x, o, OR\rangle\ \langle x, o, R\rangle^*\ \langle x, o, CR\rangle]^*.$$

In Read2, the read operations of a caller must occur between open_read and close_read operations of that caller. In contrast to Write, access here is not restricted to one object at the time. By quantification, the predicate of the trace set only considers the behavior of individual objects in the environment. Finally, Read2 refines Read.

Example 3 Consider the specification RW of an object controlling both read and write access, merging the specifications Write and Read2. Write access should be exclusive, but once a caller has obtained read access, several callers are allowed to perform read operations. The object set of RW is $\{o\}$. The alphabet of RW is obtained from Write and Read2, so $\alpha(\mathrm{RW}) \triangleq \alpha(\mathrm{Write}) \cup \alpha(\mathrm{Read2})$. RW lets objects perform read operations when granted write access. We here represent by M any event $\langle x, y, M\rangle$ where $M \in Mtd$ and define a predicate P_{RW1} on traces h to express admissible behavior as follows:

$$P_{RW1}(h) \triangleq \forall x \in Object : \\ h/x \textbf{ prs } [OW\ [W \mid R]^*\ CW \mid OR\ R^*\ CR]^*.$$

Let $\sharp(h)$ denote the length of a trace h and h/M the restriction of a trace h to the set of events $\{\langle o', o, M\rangle \mid o, o' \in Obj \wedge M \in Mtd\}$. Denote by $w(h) \triangleq \sharp(h/OW) - \sharp(h/CW)$ and by $r(h) \triangleq \sharp(h/OR) - \sharp(h/CR)$. Define $P_{RW2}(h) \triangleq (w(h) = 0 \vee r(h) = 0) \wedge w(h) \leq 1$. Now, we define the trace set of RW as

$$\mathcal{T}(RW) \triangleq \{h : \mathrm{Seq}[\alpha(\mathrm{RW})] \mid P_{RW1}(h) \wedge P_{RW2}(h)\}.$$

The specification RW refines both Read and Write from Example 1, but it does not refine Read2 from Example 2, as events reflecting Read operations may occur when read access is closed, i.e. when the calling object has write access.

4 Composition of Interface Specifications

Object composition is an abstraction mechanism encapsulating two objects, where the communication between the objects is considered *internal* to the composition. Internal communication events are hidden in the alphabet and are therefore not reflected in the (observable) traces that appear in the trace set of the composition.

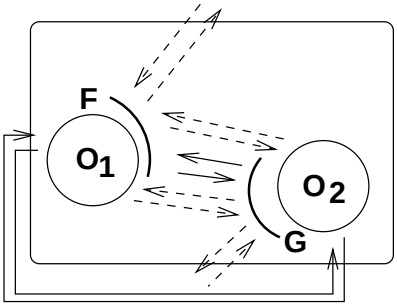

Figure 1. Composition of interface specifications F and G of objects o_1 and o_2.

Definition 3 *The set $\mathcal{I}(o_1, o_2)$ of internal events of the composition of two objects o_1 and o_2 is the set of all possible communication events between these objects, defined as*

$$\mathcal{I}(o_1, o_2) \triangleq \{\langle o_1, o_2, m\rangle \mid m \in Mtd\} \\ \cup \{\langle o_2, o_1, m\rangle \mid m \in Mtd\}.$$

Let Γ and Δ be interface specifications such that $\mathcal{O}(\Gamma) = \{o_1\}$ and $\mathcal{O}(\Delta) = \{o_2\}$. Define the internal events $\mathcal{I}(\Gamma, \Delta)$ of the composition of the two specifications as

$$\mathcal{I}(\Gamma, \Delta) \triangleq \mathcal{I}(o_1, o_2).$$

This definition of internal events is rather strong in the sense that the set of internal events of two interface specifications contains events that are internal to the *objects* specified, but the events are not necessarily in the alphabet of either of the specifications. Figure 1 illustrates how each specification only considers a subset of the alphabet of the object it describes. Apart from communication with the environment, there are events between the two objects that are known to both specifications (solid arrows), to either F or G (stapled arrows), and to neither specification (solid arrows). All events between o_1 and o_2 are hidden when F and G are composed. This definition seems necessary to ensure compositional refinement for interface specifications, due to the possibility of alphabet expansion incorporated in the notion of refinement. We denote by $\Gamma||\Delta$ the composition that encapsulates two specifications Γ and Δ, where the internal activity is hidden from an external observer.

Definition 4 *Let Γ and Δ be interface specifications. Then $\Gamma||\Delta$ is the specification $\langle \mathcal{O}, \alpha, \mathcal{T}\rangle$ where 1) $\mathcal{O} \triangleq \mathcal{O}(\Gamma) \cup \mathcal{O}(\Delta)$, 2) $\alpha \triangleq (\alpha(\Gamma) \cup \alpha(\Delta)) - \mathcal{I}(\Gamma, \Delta)$, and 3) $\mathcal{T} \triangleq \{h/\alpha \mid h/\alpha(\Gamma) \in \mathcal{T}(\Gamma) \wedge\ h/\alpha(\Delta) \in \mathcal{T}(\Delta)\}$.*

Objects are modeled using traces of structured communication events including information on both the sender and receiver of a method call. The internal activity of an object is assumed not to be observable, so it does not appear in the traces of the object. Due to object identity, the following property then holds:

Property 5 $\Gamma||\Gamma = \Gamma$, *for any interface specification* Γ.

In contrast, this is not the case for process languages without object identity, where a method call to the parallel composition of two identical processes would yield execution of the method in both copies of the process [11].

The composition of an interface specification with itself might seem contradictory to the idea of object composition. However, with multiple inheritance and aspect-oriented paradigms like ours, there may be several (partial) specifications of an object identifier. However, the composition of two interface specifications of the same object is the weakest refinement of both specifications, assembling different roles of the object into a whole.

Lemma 6 *Let* Γ_1, Γ_2 *be interface specifications of some object* o. *Then* $\Gamma_1||\Gamma_2 \sqsubseteq \Gamma_1$ *and* $\Gamma_1||\Gamma_2 \sqsubseteq \Gamma_2$. *For all specifications* Δ, *if* $\Delta \sqsubseteq \Gamma_1$ *and* $\Delta \sqsubseteq \Gamma_2$, *then* $\Delta \sqsubseteq \Gamma_1||\Gamma_2$.

Compositional Refinement of Interface Specifications. As a first approach to compositional refinement in the setting of aspect-oriented specifications, consider the composition of two interface specifications. This case is simpler than compositional refinement of component specifications because there is no hiding of internal events in either specification. In a composition of interface specifications, we want to refine one of the specifications and reason about its behavior in a context, relative to the behavior of the original specification in the same context.

Theorem 7 *Let* Γ *and* Γ' *be interface specifications of an object* o_1 *and let* Δ *be an interface specification of an object* o_2. *Then* $\Gamma' \sqsubseteq \Gamma \Rightarrow (\Gamma'||\Delta) \sqsubseteq (\Gamma||\Delta)$.

5 Components

A component encapsulates an arbitrary number of objects, hiding implementation details. In terms of observable behavior, encapsulation hides internal communication. Semantically, the alphabets and trace sets of the objects are given and unique. From these, we construct the alphabet and trace set of components. The objects of a component are explicit; we represent a component C that encapsulates a set of objects $\{o_1, \ldots, o_n\}$ simply by that set of objects. The notion of internal events is extended to components.

Definition 8 *The internal events of a set* S *of objects are defined by the pairwise union of the internal events of the objects in the set,* $\mathcal{I}(S) \triangleq \bigcup_{o,o' \in S} \mathcal{I}(o, o')$.

A component encapsulates its objects directly, without internal structure, i.e. it does not encapsulate "smaller" components. The alphabet and trace set of a component are constructed from the alphabets and trace sets of its objects.

Definition 9 *Let* C *be a component, encapsulating a set of objects* $\{o_1, \ldots, o_n\}$. *Define the set of possible observable communication events* α^C *of* C *by* $\alpha^C \triangleq \bigcup_{o \in C} \alpha^o - \mathcal{I}(C)$. *Let* $h \in Seq[\bigcup_{o \in C} \alpha^o]$. *The trace set* $\mathcal{T}^C$ *of* C *describes all possible executions of the component, defined as*

$$\mathcal{T}^C \triangleq \{h/\alpha^C \mid \bigwedge_{o \in \mathcal{O}(C)} h/\alpha^o \in \mathcal{T}^o\}.$$

Component composition becomes the union operation on sets, and is obviously commutative and associative. Due to object uniqueness, components are compositional in the sense that we can derive $\alpha^{C_1 \cup C_2}$ and $\mathcal{T}^{C_1 \cup C_2}$ directly from the alphabets and trace sets of C_1 and C_2.

6 Component Specifications

In this section, we consider the specification of (multi-object) components. In an aspect-oriented setting, we must assume that there are overlapping occurrences of the same object identifier in different specifications. Composition in this case is non-trivial and its properties must be examined.

A component specification describes a service provided by the component to its environment in terms of observable behavior. Although we do not know the details of how this service is implemented, we can construct the component's maximal set of internal events as we know its object set. However, an event that is internal in one specification Γ may be in the alphabet of another specification Δ. Were we to compose Γ and Δ, the visible behavior of Δ could restrict the (possible) internal behavior of Γ.

Definition 10 (Composability) *Two component specifications* Γ *and* Δ *are* composable *if and only if*

$$\alpha(\Gamma) \cap \mathcal{I}(\mathcal{O}(\Delta)) = \emptyset \wedge \mathcal{I}(\mathcal{O}(\Gamma)) \cap \alpha(\Delta) = \emptyset.$$

For two composable component specifications it is possible to determine whether their composition is "meaningful" by considering the observable communication traces of the two specifications. Most often, only a subset of the internal events actually take part in this activity, so composability will generally be a too restrictive condition for the composition rule. However, the criterion is statically determinable.

Boiten et al have a related notion [7]. In their work, two specifications are called *consistent* if they have a common refinement. The (least) common refinement is obtained by unification of the specifications. As the trace sets in our formalism are prefix closed, two specifications always have a common refinement, with a trace set including the empty

trace. In our setting, (non-trivial) consistency cannot be determined by external observation unless the specifications are composable.

Definition 11 *Let Γ and Δ be composable component specifications. Then $\Gamma||\Delta$ is the specification $\langle \mathcal{O}, \alpha, \mathcal{T} \rangle$ where 1) $\mathcal{O} \triangleq \mathcal{O}(\Gamma) \cup \mathcal{O}(\Delta)$, 2) $\alpha \triangleq \alpha(\Gamma) \cup \alpha(\Delta) - \mathcal{I}(\mathcal{O})$, and 3) $\mathcal{T} \triangleq \{h/\alpha \,|\, h/\alpha(\Gamma) \in \mathcal{T}(\Gamma) \wedge \ h/\alpha(\Delta) \in \mathcal{T}(\Delta)\}$.*

The following properties are due to composability.

Property 12 *The composition of specifications is commutative and associative.*

Soundness for specifications in general is understood as an extension to the notion of soundness for interface specifications in Section 2, relating the traces of components to those of component specifications. A naive generalization of Lemma 6 is not straightforward, as hiding of internal events implies that $\alpha(\Gamma) \not\subseteq \alpha(\Gamma||\Delta)$ for specifications Γ and Δ. However, the following lemma ensures soundness for the composition of specifications of a given component.

Lemma 13 *Composition preserves soundness, i.e. if Γ and Δ are sound specifications of a given component C, then $\Gamma||\Delta$ is a sound specification of C.*

The refinement of component specifications differs from the special case of interface specifications previously considered. For the refinement Γ' of a component specification Γ, we are allowed to introduce new objects, so we cannot expect $\mathcal{I}(\Gamma) = \mathcal{I}(\Gamma')$ to hold. In order to obtain reasoning control over the refinement of component specifications, we need one further restriction, motivated as follows: Consider the composition of two specifications Γ and Δ. Say that Γ' is a refinement of Γ, and Γ' includes an event that is in the alphabet of Δ and therefore visible in the composition $\Gamma||\Delta$. Then this event should also be visible in $\Gamma'||\Delta$. This might occur due to the possible inclusion of new objects in the refinement of Γ. No restriction is needed for the inclusion of new methods for objects that already occur in Γ.

Definition 14 (Propriety) *Let Γ, Γ' and Δ be component specifications such that $\Gamma' \sqsubseteq \Gamma$. Let α' denote the set of events that involve objects of Γ' but not of Γ, so*

$$\alpha' \triangleq \{\langle o_1, o_2, m\rangle \,|\, (o_1 \in \mathcal{O}(\Gamma') \vee o_2 \in \mathcal{O}(\Gamma')) \wedge o_1 \notin \mathcal{O}(\Gamma) \wedge o_2 \notin \mathcal{O}(\Gamma)\}.$$

Then Γ' is a proper refinement of Γ with respect to Δ if $\alpha' \cap \alpha(\Delta) = \emptyset$.

This indicates that new objects can be included in Γ only if these are not in the communication environment of Δ. We can interpret the propriety criterion as an interdiction to reduce the communication environment of a composition of component specifications when we refine one of them. For specifications satisfying the propriety criterion, the following lemma holds.

Lemma 15 *Let Γ' and Δ be composable specifications and let Γ' be a proper refinement of a specification Γ with respect to Δ. Then $(\alpha(\Gamma) \cup \alpha(\Delta)) \cap \mathcal{I}(\mathcal{O}(\Gamma'||\Delta)) = (\alpha(\Gamma) \cup \alpha(\Delta)) \cap \mathcal{I}(\mathcal{O}(\Gamma||\Delta))$.*

From this lemma, we infer compositional refinement when propriety and composability are assumed.

Theorem 16 *Let Γ, Γ' and Δ be component specifications such that 1) Γ' is a proper refinement of Γ with respect to Δ and 2) Γ' and Δ are composable. Then $\Gamma'||\Delta \sqsubseteq \Gamma||\Delta$.*

Remark that if no new object identifiers are introduced in a refinement step, propriety and composability are always preserved by refinement.

7 Examples

We illustrate by examples some issues concerning composition and refinement in our formalism. For simplicity, only interface specifications are considered here. Using projection, we avoid some difficulties related to composing specifications at different levels of abstraction (Example 4). However, this flexibility lets us introduce new deadlock situations in a refinement step (Example 5).

Example 4 Let WriteAcc modify Write of Example 1, so that only the object c makes calls to the methods of o, the object specified by WriteAcc. (This is done by modifying the predicate of the trace set, so WriteAcc refines Write.) We now specify by Client a write client making calls to the write access controller o. The Client object c calls the write method of o and then confirms these calls to a monitor object o' in its environment. Client has the object set $\mathcal{O}(\text{Client}) = \{c\}$ and the alphabet:

$$\alpha(\text{Client}) \triangleq \{\langle c, x, W(d)\rangle \,|\, x \in Objects, d \in Data\} \cup \{\langle c, x, OK\rangle \,|\, x \in Objects\}.$$

Let Reg be a regular expression which ignores the written data values, so $Reg = \langle c, o, W(\_)\rangle \ \langle c, o', OK\rangle$. The trace set of Client is

$$\mathcal{T}(Client) \triangleq \{h \in \text{Seq}[\alpha(\text{Client})] \,|\, h \ \textbf{prs} \ Reg^*\}.$$

Now compose Client with WriteAcc. Without projection, this composition results in an immediate deadlock as OW is not in the alphabet of Client, so $\mathcal{T}(Client||WriteAcc)$ would only include the empty trace. With projection, this is not the case. Let $t \in \text{Seq}[\alpha(\text{Client}) \cup \alpha(\text{WriteAcc})]$ satisfy t **prs** $[\langle c, o, OW\rangle \ Reg \ \langle c, o, CW\rangle]^*$. Then $t/\alpha(\text{Client}) \in \mathcal{T}(Client)$ and $t/\alpha(\text{WriteAcc}) \in \mathcal{T}(WriteAcc)$. Therefore, when internal events are hidden in the composition, we only see calls to o':

$$\mathcal{T}(Client||WriteAcc) = \{h : \text{Seq}[\alpha(\text{Client}||\text{WriteAcc}) \,|\, h \ \textbf{prs} \ \langle c, o', OK\rangle^*\}.$$

In Example 4, we show how specifications at different levels of abstraction can be composed without necessarily causing deadlock. Without the use of projection, the composition of Client and WriteAcc would yield a deadlock. In that case, we could refine Client into a specification Client′, include OW and CW operations at appropriate places in the traces, and thus remove the deadlock through refinement: observable events would start to appear in the composition of Client′ and WriteAcc. This is clearly undesirable. Using projection as we do, Client appears as more abstract than Client′ and WriteAcc because it ignores certain communication details. We can still compose the specifications and Client′||WriteAcc has the same trace set as Client||WriteAcc in our formalism.

However, the use of projection does not exclude deadlock. In Example 5, we show how deadlock can be introduced in a refinement step.

Example 5 Let Client2 refine Client, with the same object set. Introduce the method OW of the object o in the specification, so $\alpha(\text{Client2})$ becomes $\{\langle c, o, OW\rangle\} \cup \alpha(\text{Client})$. In the trace set of Client2, we let the OW event appear after write events W, i.e. in the opposite order of how they appear in WriteAcc, so the trace set of Client2 is defined by

$$\mathcal{T}(Client2) \triangleq \{h \in \text{Seq}[\alpha(\text{Client2})] \mid h \textbf{ prs } [Reg\ \langle c, o, OW\rangle]^*\}.$$

The trace set of Client2||WriteAcc only contains the empty trace, as the first event in a trace of WriteAcc is $\langle c, o, OW\rangle$, whereas for a trace of Client2, it is $\langle c, o, W(d)\rangle$. Hence, Client2||WriteAcc trivially refines Client||WriteAcc.

8 Discussion

We have proposed a formalism supporting partial specifications of objects with identity, and proven a compositional refinement property for this formalism. Arguments for aspect-oriented system design can be found in the literature [7, 12, 15]. Aspect-oriented specification has been studied in the context of multi-paradigm specification techniques, where it is assumed that the different notations can be translated into a common (state-based) framework [6, 7, 26]. In these approaches, emphasis is on composition without hiding, i.e. on unifying the specifications in a (sufficiently large) state space.

For open distributed systems, we prefer to think of objects and components as black boxes, where implementation details are unavailable for reasoning. Hence, we use a trace-based approach where composition hides internal activity in the tradition of process languages like CSP [11] and FOCUS [9]. However, we use explicit object identities in the communication traces, rather than communication channels, and thereby avoid inherent difficulties of channel-based approaches [24, 25]. The formalism we propose is compositional, as it suffices to consider externally available information of specifications in order to reason about composition and refinement. (Internal events are not known, but we construct the worst-case scenario from the available object identities.) The formalism is augmented with further syntactic coating in the specification language OUN [20].

Traditionally, refinement relations compare specifications with fixed alphabets [1, 5, 9, 23], although some approaches propose to decompose events, with mappings between different alphabets [16]. In an aspect-oriented setting, there are good reasons to allow alphabet expansion in refinement steps. It gives us multiple inheritance of behavior: two specifications can have a common refinement. Also, we capture component upgrade by addition of new functionality, in a manner which resembles subclassing in object-oriented languages. Traditional refinement appears as a special case.

Compositional refinement in a framework for partial specifications seems novel to this paper. Partiality and object identity add difficulty to the problem, as events related to one specification may be included in the alphabet of another through refinement. The framework for partial specifications used in this paper, has been encoded in the Prototype Verification System (PVS) [21] and compositional refinement (Theorem 16) and related properties have been verified in the PVS theorem prover. For concrete specifications, we find that trace sets are specified by means of predicates, so proof of correct development steps are most often discharged by the theorem prover in a straightforward manner.

For simplicity, liveness has not been considered in this paper. In future work, we would like to examine how specifications with liveness properties would behave in our setting. As demonstrated by the examples of Section 7, our use of projection offers a certain flexibility which lets us avoid some deadlock situations in composition, but it also lets us introduce potential deadlock situations in refinement steps. Liveness reasoning in this setting will therefore lead to an interesting extension of the results presented in this paper.

9 Conclusion

Object-orientation and multiple viewpoints are often recommended for the development of open distributed systems. In this paper, we consider the composition and refinement of partial specifications of object-oriented components. For this purpose, a formalism for object-oriented specifications is introduced, based on finite traces of communication events reflecting remote method calls between object identifiers. Partial specifications represent viewpoints or aspects of objects.

We propose a composition rule encapsulating objects and a refinement relation which allows alphabet expansion and

introduction of new objects in a refinement step. In this formalism, we study compositional refinement properties for partial specifications of object identifiers. Several specifications may describe the same object.

Compositional refinement properties have previously been studied in the literature, but formalisms supporting multiple viewpoints have not received much attention. The paper demonstrates how an aspect-oriented specification style and explicit object identities lead to additional difficulties and suggests a solution with sufficient conditions to obtain compositional refinement in this setting. If we accept these restrictions, we get an aspect-oriented specification notation which supports stepwise refinement of component specifications, using the proposed refinement relation. The notation can be extended with a syntactic coating to provide an easy-to-use specification language for behavioral interfaces.

The paper demonstrates how an aspect-oriented approach to object-oriented system development necessitates extra caution due to explicit object identities and encapsulation. Nevertheless, we think the approach seems promising for the development of open distributed systems, with its focus on the specification of system services by observable behavior.

Acknowledgment

This work is part of the ADAPT-FT project. The authors are grateful for the helpful comments of Isabelle Ryl, Ketil Stølen, and Peter C. Ölveczky.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[2] G. A. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, Jan. 1997.

[3] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, Oct. 1985.

[4] K. R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Systems*. Springer, Berlin, 1991.

[5] R. J. R. Back and J. von Wright. Trace refinement of action systems. In B. Jonsson and J. Parrow, editors, *Proc. 5th International Conference on Concurrency Theory (CONCUR'94)*, LNCS 836. Springer, Uppsala, Aug. 1994.

[6] L. Blair and G. Blair. Composition in multi-paradigm specification techniques. In R. Ciancarini, A. Fantechi, and R. Gorrieri, editors, *Proc. 3rd International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*. Kluwer, Feb. 1999.

[7] E. Boiten et al. Consistency and refinement for partial specification in Z. In M.-C. Gaudel and J. Woodcock, editors, *Proc. 3rd International Symposium of Formal Methods Europe (FME'96)*, LNCS 1051. Springer, Mar. 1996.

[8] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, USA, 1999.

[9] M. Broy. Compositional refinement of interactive systems. *Journal of the ACM*, 44(6):850–891, Nov. 1997.

[10] O.-J. Dahl and O. Owe. Formal methods and the RM-ODP. Research Report 261, Dept. of informatics, Univ. of Oslo, 1998.

[11] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[12] ITU Recommendation X.901-904 ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model*. ISO/IEC, July 1995.

[13] E. B. Johnsen and O. Owe. Composition and refinement for partial object specifications (full version). Research Report 301, Dept. of informatics, Univ. of Oslo, 2002.

[14] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74: Proc. IFIP Congress 74*. North-Holland, Aug. 1974.

[15] G. Kiczales et al. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proc. 11th European Conference on Object-Oriented Programming (ECOOP'97)*, LNCS 1241. Springer, New York, NY, June 1997.

[16] B. H. Liskov and J. M. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, Nov. 1994.

[17] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, July 1981.

[18] E. K. Nordhagen. *Divide et Impera: A Computational Framework for Verifying Object Component Substitutability*. PhD thesis, Dept. of informatics, Univ. of Oslo, 1998.

[19] Object Management Group. The common object request broker: Architecture and specification. OMG, 1999.

[20] O. Owe and I. Ryl. OUN: a formalism for open, object oriented, distributed systems. Research Report 270, Department of informatics, University of Oslo, 1999.

[21] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE'92)*, LNCS 607. Springer, Saratoga, NY, June 1992.

[22] D. L. Parnas and Y. Wang. The trace assertion method of module interface specification. Technical Report 89-261, Department of Computing and Information Science, Queen's University, Kingston, Oct. 1989.

[23] W. A. Roscoe. *Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[24] N. Soundarajan. Communication traces in the verification of distributed systems. In *Proc. 2nd Northern Formal Methods Workshop*, Ilkley, UK, 1997.

[25] J. Widom, D. Gries, and F. B. Schneider. Completeness and incompleteness for trace-based network proof systems. In *Proc. 14th ACM Symposium on Principles of Programming Languages (POPL)*, 1987.

[26] P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions of Software Engineering and Methodology*, 2(4):379–411, Oct. 1993.

[27] J. Zwiers. *Compositionality, Concurrency and Partial Correctness*, LNCS 321. Springer, 1989.