T U M

I N S T I T U T F Ü R I N F O R M A T I K

Towards Service-Based Systems Engineering: Formalizing and mu-Checking Service Descriptions

Bernhard Schaetz

TUM-I0206
Juli 02

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-07-I0206-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Towards Service-Based Systems Engineering: Formalizing and μ-Checking Service Descriptions

Bernhard Schätz

Fakultät für Informatik, TU München
schaetz@in.tum.de

Abstract. Using graphical description techniques for the modeling of distributed reactive systems is especially useful in the early phases like requirements engineering. There, often a trial-and-error approach is applied to develop a first system specification. Thus, even there a precise meaning of those description techniques is necessary to check for consistency and completeness of a service-oriented description of the requirements. We introduce a formal semantics for the scenario-based graphical language Chisel. We show how this formalization can be directly applied to model checking service specifications without the need for an explicit operational system model thus supporting the development of complete and consistent service descriptions.

1 Introduction

Using services as basic concept eases the specification of reactive systems with a high degree of interaction with its environment as found, e.g., in telecommunication or modern automotion combining services like ABS and ABC. This approach allows to break up the complex system functionality into smaller modules, and to describe each module as a coherent part of the behavior of the system. This modularity supports a more manageable and comprehensible description of the system functionality. Those behavioral modules are generally called services – or, especially in the field of telecommunication, features.

To structure a specification like this, UML use cases, combined with collaboration diagrams or sequence diagrams can be used. While the use cases describe the interface between system and environment and the overall structure of the services, each sequence or collaboration diagram describes the interaction taking place during the execution of a single service. Of course, also other graphical representations of services can be used. Telecordia/BellCor introduced Chisel as a graphical notation to describe telecom features ([AG+98]). Originally introduced informally, the syntax was formally described in [Tur00], leaving out a formal semantics. Informally, each diagram, as described in [GB+99], represents a behavior of the system as a decision tree, describing a possible course of actions. The left side of Figure 1 shows the Chisel representation of a part of the basic telephone service. In this article we will describe a methodical treatment of service-based specifications using Chisel diagrams. However, this approach is not Chisel specific. It carries over to other notations describing a course of actions like with (high level) sequence diagrams.

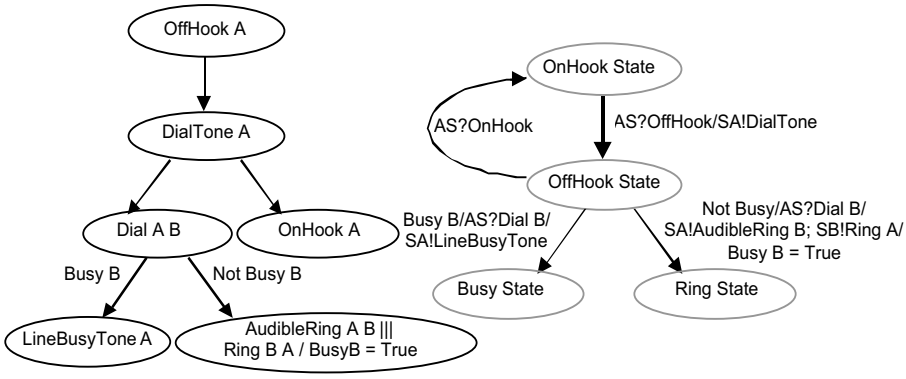

Figure 1: Elements of a Chisel Diagram and its Automaton Representation

To describe a system with services enhancing the plain ordinary telephone system (POTS), a collection of Chisel diagrams is used. Each diagram describes an additional service by extending the original POTS diagram to describe features like Terminating Call Screening (TCSC) or Call Forward on Busy Line (CFBL). However, in general those services are not independent of each other. Thus, when combined to form a complete system description, they influence each other, resulting in feature interaction. In the worst case, the combined services may be incompatible, resulting in an inconsistent specification. Thus, the ability to check a specification for inconsistency is needed. Since feature interaction detection is a problem of the requirements phase and thus often occurs during trial-and-error based requirements definition, especially automatic proof techniques are of interest.

To automatically detect possible interactions we need a precise meaning of those diagrams. Besides making limitations of the informal semantics obvious (e.g., disallowance of simultaneous actions) and identifying incompleteness in the specification (e.g., unspecified reaction on input), applying model checking to the formal description of the features allows us to automatically detect cases of unwanted interaction without any bias of implementation.

1.1 Methodical Aspects of Service Interaction

As mentioned above, in a service-based engineering process, services are used to break up the complex functionality of a system into smaller parts. Naturally, each service froms only a *partial description* of the behavior of the complete systems. To obtain the complete description of the behavior of the system, services must be recombined. Thus, while supporting a modular and often easier-to-understand description of the system, services-based modeling of a system must deal with the issue of combining services.

For a good formalization of a description technique we have to look at the operations and methods that are applied by the users of this technique and find a direct formal correspondence of those operations and methods ([Zav99]):

> "(The) system specification ... takes the form B+F1+F2+F3 ..., where B is a base specification, each Fi is a feature module, and + denotes some feature-composition operation."

"A feature interaction is some way in which a feature or features modify or influence another feature in defining overall system behavior. (..)"

"A bad feature interaction is one that causes the specification to be incomplete, inconsistent, or unimplementable, or that causes the overall system behavior to be undesirable."

Thus, for a direct correspondence of our formalization we have to check that

The formalization interprets diagrams independently with the possibility to interact.

The formalization of feature combination is a simple combination of those features.

A feature-interaction problem is an inconsistent or incomplete combination of features.

We do not generally consider incompleteness as undesirable but as underspecification supporting further refinement in later steps. Since, however, the detection of underspecification is of methodical importance, a way to deal with underspecification in our approach is described in Sections 2.3 and 2.6. [JZ00] shows that the question whether an interaction is undesirable is rather complex leading to hints for a suitable resolution at best. Thus, we concentrate on a simple formalization of feature interaction detection supporting the feature designer based on well-known results from I/O-Automata, TLA ([Lam93]) and relational μ calculus ([Par76]) by automation of the requirements engineering process.

Analytical approaches support the detection of feature interaction problems by offering mechanized techniques to check for interactions (e.g., Lutess/Lesar [dB99]). One major drawback so far is the bias of implementation: model checkers like SMV are used to validate whether temporal properties hold for a given system described by a state machine. Generally, in those approaches feature descriptions are transformed into two different forms of descriptions: the expected functionality of a feature expressed by a temporal logic property and a description of a state machine describing a part of the abstract implementation of the system in question. Then, interaction is checked for by combining the state machine parts in one machine implementing the features in an abstract system and verifying whether this implementation respects the described properties. But combing services within an abstract implementation may lead to an implicit resolution of conflicts; this implementation bias may hide interactions defined by a scenario-based description. We describe an approach without this double interpretation of features and its bias. Instead of creating an abstract implementation, our approach directly checks for the compatibility of the requirements of the services.

1.2 Example: Screening and Forwarding

In [GB+99], additional services extending POTS are described by replacing or extending states and transitions of the POTS diagram to form new Chisel diagrams. To demonstrate our approach, we will use the following two services extending the basic telephone service:

- Terminating Call Screening (TCSC): A subscribing user can prohibit calls from other users adding their terminals to a Screen List. Calls from screened terminals are not announced at the callee; the caller is informed by a corresponding message.
- Call Forwarding on Busy Line (CFBL): A subscribing user can redirect calls to a third party if a call occurs while his terminal is busy.

Combining those two services can lead to different forms of feature interaction:

- If user B subscribes to both CFBL with Forward C and TCSC with a Screen List containing A, what happens if A calls B while B is busy? Should A be forwarded resulting in a ring tone played to A or screened resulting in a screening message played to A?

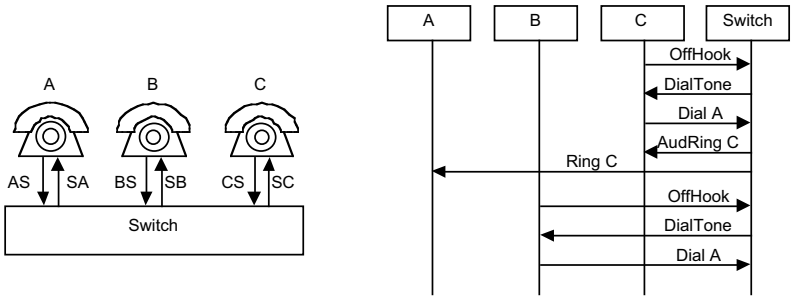

Figure 2: Switch with three terminals and a possible interaction of features

- If A subscribes to TCSC with a Screen List containing B and C subscribes to CFBL with Forward A, what behavior will occur if B calls C while C is busy? Will B be forwarded to A violating the screening of B, or will B be screened violating the forwarding?

2 Formal approach

To make Chisel diagrams directly accessible to automatic detection of service interaction, we need a formalization of a collection of diagrams. Thus, we

- define a *system model* describing how a system interacts with is environment
- define a *mapping* of Chisel diagrams onto the system model
- define the meaning of a *combination* of services described by Chisel diagrams
- define when an *interaction* of services occurs

Furthermore, we will also extend the approach to methodically treat underspecification and

- define when a combination of services is *complete*

The following subsections discuss those steps in detail. Furthermore, the application of those steps in combination with model checking is explained.

2.1 The System Model

Since services are used to structure the requirement specification of a system we first have to characterize the possibilities of interactions between system and environment. Generally, this interface is described by means of communication channels, as found, for example, in SDL. Figure 2 shows a graphical representation of a telephone system consisting of three terminals (A, B, C), a switch, and their connecting channels as used in the TCSC/CFBL example.

The sketch of the meaning of Chisel ([GB+98]) suggests interleaving trace semantics as system model, i.e., model of system behavior. There, a system execution corresponds to a sequence of communication actions[1] like *OffHook* (lifting the receiver), *DialTone* (playing a dial tone), *Dial B* (dialing the address of B), or *Ring A* (announcing a call from A). Here we use a relational semantics; a system execution corresponds to a sequence of tuples of actions,

[1] We leave out the first address (like A in OffHook A) since those actions are assigned to a channel.

each element describing a message sent via a single channel. This model supports the notion of concurrently occurring events. This is favorable since

- the description of services makes use of arbitrary interleaving of events to simulate concurrent events, which is simplified in the concurrent model;
- the interleaving model ignores cases of concurrent events due to expressiveness which may however arise in a real implementation;
- the automaton-based formalization can be simplified since no explicit treatment of fairness is needed (as, e.g. with I/O-Automata [LT89])

First, we introduce the notion of an interface. A system (or component) communicates with its environment via unbuffered channels. While messages can be sent simultaneously via different channels, they are sequentialized if transferred via the same channel. The type of a channel is described by the set of messages sent over it. With POTS, this is the set of the above messages including *OffHook*, *DialTone*, as well as a *Dial* and a *Ring* message with an associated address. For a detailed description of the system model see [HSE97]. An event of an execution may consist of several transmitted messages if transferred via different channels. It is formalized as a tuple, each tuple element representing a message sent over the corresponding channel (or *Nil* if none was sent). We define a system execution to take place in rounds and model it by an infinite trace of events. Thus the nth element of a trace corresponds to the values of the event sent during the nth round of the system execution. We only accept input closed sets as system behavior descriptions. Here, this is defined as

$$\forall r, s.\ r \circ s \in T. \exists i_1, \ldots i_m. \forall o_1, \ldots o_n, t.\ r \circ (i_1, \ldots i_m, o_1, \ldots o_n) \circ t \in T \qquad (1)$$

This corresponds, e.g., to input enabledness of I/O Automata.

The behavior of a system with input channel types $I_1, \ldots I_m$ and output channels types $O_1, \ldots, O_n$ is defined as an input closed set T of traces of events with

$$T \subseteq (I_1 \cup \ldots \cup I_m \cup O_1 \cup \ldots \cup O_n)^{\infty}$$

In our example, we have three input channel types I_{AS}, I_{BS}, I_{CS}; they consist of the set of messages $\{OffHook, Dial\ A, Dial\ B, Dial\ C, \ldots\}$. Correspondingly, there are three output channel types O_{SA}, O_{SB}, O_{SC} of set $\{DialTone, Ring\ A, Ring\ B, Ring\ C, \ldots\}$

2.2 Formalizimg Service Descriptions

A diagram is mapped onto the system model by transforming it into a corresponding automaton. As shown in Figure 1, a diagram describes a tree of executions triggered by actions of the environment (*OffHook*, *Dial B*); depending on those actions, output actions (*DialTone, Ring A*) and assignments to variables (*BusyB = True*) are performed. Alternative executions are started by branching on different input events (*Dial B* vs. *OnHook*) or different values of variables (*BusyB*, *Not BusyB*). In case of simultaneous output actions or assignments, arbitrary sequencing (|||), is used. Thus, a diagram can be interpreted as a sequence diagram formalized by an automaton. Note that the automaton does not describe a system or component behavior but a partial behavior of such a system or component.

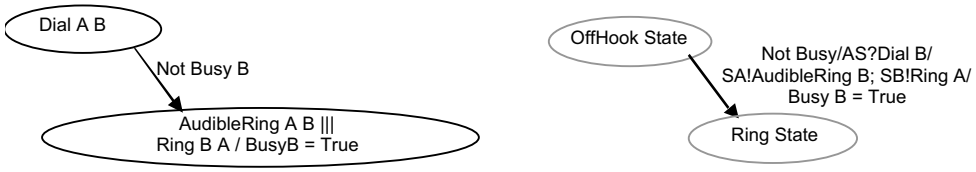

Figure 3: Chisel transition and Automaton Representation

To build the automaton, control states (*OnHook State*, *OffHook State*, *Busy*, *Ring*) are introduced denoting positions in the diagram where input events occur. Each group of input events, possibly branching because of variable values, output events, and variable assignments, is combined into a single transition, as seen in Figure 3. We use a notation described in [HSE97] with precondition (*Not BusyB*), input pattern (*AS?Dial B*; *Dial B* received on channel *AS*), output pattern (*SA!AudibleRing B;SB!Ring A*; *AudibleRing B* sent on channel *SA* while *Ring B* sent on channel SB), and postcondition (*BusyB = True*). The set of traces generated by the transition defines the requirements specification of the service described by its diagram. While a diagram represents only one call, a system characterized by diagrams accepts an arbitrary sequence of such calls. Thus, the final states of the automaton are identified with initial states, allowing a repetition of a service during system execution. Figure 1 shows an example of such a feedback loop triggered by the *OnHook* signal.

To formalize such an automaton we use a relation typed similar to the interface of the system (i.e. the tuple of input and output channels as in Section 2.1). However, a feature on the one hand considers only a part of the interface, while on the other it depends on variables like the control state. Thus, the corresponding relation comprises a subset of the channels, the control and data state (i.e. all variables) before and after the transition. In case of POTS the type of the control state is $C = \{OnHook\ State, OffHook\ State, Busy, Ring, \ldots\}$ and a data state consists of the Boolean variable BusyB. Thus this transition relation is typed

$$F \subseteq C \times B \times I \times I \times O \times O \times B \times C$$

with B denoting Boolean values, I the input, and O the output channel type.[2] Since this feature considers two parties (initiating/called party) two input and two output lines are needed.

A transition is formalized as the conjunction of pre- and postcondition as well as the channel predicates. Channel predicates are simply the equality between the channel variable and the value assigned to or read form the channel as described by the channel patterns. Thus, for the transition described above with a formal parameter list of

$$State\ C; BusyB\ B; AS\ I; BS\ I; SA\ O; SB\ O; BusyB'\ B; State'\ C$$

we obtain the formalization

$$State = OffHook \wedge \neg BusyB \wedge AS = Dial\ B \wedge$$
$$SA = AudibleRing\ B \wedge SB = Ring\ A \wedge BusyB' = True \wedge State' = Busy$$

[2] Note that in the example all input channels carry the same messages as well as the output channels.

with *State* and *State'* as well as *BusyB* and *BusyB*' denoting the control and data state, resp., before and after the transition. The transition relation is constructed via the disjunction of the formalization of each transition.

2.3 Treating Underspecification

Since the system model requires input-closure, the transition relation must be input enabled, i.e., for each state and possible set of input messages there must be a corresponding default transition. While a requirement specification deals with a complete system including all input and output channels, a Chisel diagram is restricted to events relevant to the service leaving all other aspects underspecified. To deal with underspecification, we treat underspecification as chaotic non-determinism allowing arbitrary behavior. This form of interpretation offers two advantages:

- When translating Chisel diagrams into relational μ calculus, underspecified behavior automatically leads to underspecification.
- Since services specify partial behavior, this interpretation allows a simple formalization of the combination of service descriptions, as described in the following subsection.

Formally, this treatment of underspecification is achieved by disjunctively adding a default clause leaving output and resulting control and data state unspecified for all states and inputs not accounted for by the specified transitions.[3] This default clause consists of a conjunction of negations of the preconditions and input patterns of all transitions for each control state. Thus, for the above transition, the conjunction is extended by

$$State = OffHook \wedge \neg(\neg BusyB \wedge AS = Dial\ B)$$

If formalized in the system model as described before, underspecification ist treated analogously to nondeterministic behavior. However, unspecified behavior in an implementation is generally a source of possible unwanted reactions (especially in embedded systems). Therefore, underspecification deserves a different treatment than explicit specification of nondeterminism for methodical reasons, for example

- detection of underspecification: detection of input or pattern where no behavior is defined for
- canonical generation of fully determined behavior: generation of a standard behavior like exception handling for undefined input pattern

Technically, the explicit treatment of underspecification is realized by a canonical extension of all data types by an additional element $\perp$ not contained in the set of possible values of a data type. Thus, for example, the type I of the input channels gets extended to $I_{\perp} = I \cup \{\perp\}$; the other types are treated accordingly. Since all expressions evaluated in the term language of the – unextended - data types have a value different from $\perp$, this mechanism can be used to detect underspecified transition relations. This is due to the fact that we are using a loose interpretation of the feature specification. Thus, when constructing the transition relation using this extension, the transition relations is closed under the property

[3] For a complete description of the formalization see [HSE97].

$$\forall c,c' \in C_\perp, d,d' \in D_\perp, i \in {I_\perp}^m, o_1,\ldots,o_n \in O_\perp .$$
$$(c,d,i,o_1,\ldots,\perp,\ldots,o_n,c',d') \in R \Rightarrow \forall v \in O.(c,d,i,o_1,\ldots,v,\ldots,o_n,c',d') \in R$$

assuming a transition relation R of type

$$R \subseteq C_\perp \times D_\perp \times {I_\perp}^m \times {O_\perp}^n \times D_\perp \times C_\perp$$

with control and data state types C and D and types I and O for the m input and n output channels.[4] Intuitively, this property relates underspecification with nondeterminism: in case the transition relation does not (completely) specifiy the reaction of the system by leaving some output undefined for a certain input pattern, any output is legal for all substitution of the undefined input (i.e. for all possible values of the same data types in the place of $\perp$). Besides the extension of the input and output types of the relation, also the control and data space of the relation is extended canonically. Technically, the set of control and datea states is extended analogously with a $\perp$ element, denoting the state reached in case of an underspecified transition. Thus a similar closure property holds for the control space of the transition relation:

$$(c,d,i,o,\perp,d') \in R \Rightarrow$$
$$\forall v \in C_\perp.(c,d,i,o,v,d') \in R$$

as well as

$$(c,d,i,o,c',\perp) \in R \Rightarrow$$
$$\forall v \in D_\perp.(c,d,i,o,c',v) \in R$$

This closure property corresponds to the fact that in case no behavior is defined for a certain input, the system may change to the undefined control or data state $\perp$.[5] For the undefined state, a closure property concerning the further behavior holds:

$$(\perp,d,i,o,c',d') \in R \Rightarrow$$
$$\forall \hat{i} \in {I_\perp}^m, \hat{o} \in {O_\perp}^n, \hat{c} \in C_\perp, \hat{d} \in C_\perp.(\perp,d,\hat{i},\hat{o},\hat{c},\hat{d}) \in R$$

as well as

$$(c,\perp,i,o,c',d') \in R \Rightarrow$$
$$\forall \hat{i} \in {I_\perp}^m, \hat{o} \in {O_\perp}^n, \hat{c} \in C_\perp, \hat{d} \in C_\perp.(c,\perp,\hat{i},\hat{o},\hat{c},\hat{d}) \in R$$

Intuitively, these properties express the fact that after having reach an undefined (control or data) state, the system may behave arbitrarily for all future steps: any output may be produced for any input in the following step, again leading to the undefined state. Thus, the loose interpretation of the transition relation leads to some form of chaotic completion as, for example, found in the TCSP approach [Hoa85]. The major advantage of this interpretation is

---

[4] In the following we will drop the explicit notation of the types of variables if obvious from the context.

[5] Generally, the data space will be a complex product space. In that case, the closure property holds for each element of the product space respectively.

Subscriber	Service	Instance: Originator, further parties	Parameters
A	TCSC	B, A	Screen List = {B}
A	TCSC	C, A	Screen List = {B}
A	CFBL	B, A, C	Forward = C
A	CFBL	C, A, C	Forward = C
B	POTS	A, B	-
B	POTS	C, B	-
C	CFBL	A, C, B	Forward = B
C	CFBL	B, C, B	Forward = B

Table 1: Service Instantiations for the TCSC/CFBL Interaction

simple notions of refinement as well as composition of specifications as used in the following sections. By additionally explicitly distinguishing between underspecification and nondeterminsm we have a further methodical advantage when dealing with partial specifications as discussed in Section 2.6.

Based on this formalization we can now give a precise definition of the notion of a service. Given a system interface in form of the types of its input and output channels, a – possibly partial – input closed set of execution traces is called a service. Definition 3 formalizes this notion. Note that according to those definitions, a service differs from a system behavior only in the explicit marking of the partial behavior; a system behavior can be obtained from a service or a combination of services by dropping this explicit marking, e.g., by removing all traces containing $\perp$. Due to the closure properties defined above, removing those traces does no influence the input closure of a service.

2.4 Combining Services

Terminals (e.g., telephones) subscribe to POTS or service like CFBL. To check for feature interaction we use different *configurations* assigning services to terminals. However, a service does not describe the complete behavior of a terminal. In our example, it rather describes a scenario consisting of a call originator and at least one other participant (e.g., the called party). A diagram describing POTS as in Figure 1 is incomplete since it describes the behavior of the system only in case party A is calling party B. Therefore, we have to combine several service instances to obtain a behavior for a terminal. Furthermore, Chisel diagrams are patterns of interactions: the participating parties (like terminal A and B) occurring in a diagram are place-holders for terminals of a system. To form a system description those services must be instantiated. A possible configuration for a network of parties A, B, and C with A subscribing to TCSC with a Screen List containing B and C subscribing to CFBL with Forward B is obtained by using the service instantiations as shown in Table 1, ignoring the grayed-out entries. Since B is not subscribing to any extended service, B is subscribing to POTS by default. We need instances of POTS with called party B and callers A, B, and C to describe the behavior of a system with B subscribing to POTS. We may even decide to leave out specific instances of POTS, like the grayed-out line containing B calling B, resulting in a system description with a loop-call of B having no restricted behavior. Due to our interpretation of underspecification even uncompleted configurations can be used to check for incon-

sistency of services. If, as in that case, a combination of input actions is not dealt with by the instances of the services – like terminal B calling itself – no behavioral restrictions are defined resulting in an undefined behavior with no service interaction problem occuring. For a more complete operational description of the system the system specification must be refined by adding additional cases.

To create configuration where a terminal subscribes to more than one service, additional instances are added to the configuration. By adding the first two grayed-out lines shown in Table 1 we define a configuration with A additionally subscribing to CFBL with forward C.

As mentioned in Section 2.2, services are requirements about the system behavior and are thus interpreted as partial requirement specifications. We interpret those partial requirements universally: each observable behavior of the system must respect the service specification.[6] The loose interpretation of services as defined in subsection 2.2 is essential for the combination of services since this interpretation does not implicitly restrict unspecified behavior. Thus, services are simply combined by constructing the intersection of the sets of system behaviors described by the corresponding Chisel diagrams. On the automaton level this is equivalent to constructing the product automaton. Effectively, this is done by a simple conjunction of the transition relations using relational μ calculus.

As in the case of a single service, we define the system specification by a transition relation for the complete system. Thus, the type of this relation is defined by the product of

- all variables used by services for each instance (e.g., *BusyA*, *BusyB* and *BusyC* for the terminals A,B, and C), representing the system state prior to the transition
- all input channel variables as defined by the system interface
- all output channel variables as defined by the system interface
- all variables used by services for each instance representing the state after the transition

Thus, the system relation is dependent on the service instances used in the configuration.

Using a loose interpretation of underspecification is essential for the combination of services into a system specification. But for the validation of the system requirements generated by the combination of the services such a loose interpretation is not always adequate: similar to TLA ([Lam93]), variables or output channels not explicitly treated in a step of the transition relation are assigned arbitrary values during that step. Thus, if input occurs with no matching defined behavior, any behavior is possible. This is operationally not satisfactory: if an input action occurs not accounted for by the service instances, the system may reach a state not accessible by non-spontaneous variable change. Since this leads to counterintuitive examples of service interactions, an operational restriction similar to the TLA approach is applied to the combined specification: state variables (i.e. both control and data state) with undefined values remain unchanged while undefined output values are defined to be Nil.

2.5 Formalizing Service Interaction

As mentioned in Subsection 1.1, we define a service interaction problem to occur if the behavior defined by the combined services is inconsistent. Actually, since here interaction boils down to inconsistency, from a formal and methodical point of view, there is no need to re-

[6] For the universal and other forms of interpretations of specifications, e.g. existential, see [SH99], or, in the context of Message Sequence Charts, [Krü00].

strict interaction to a combination of features. In cases where simple diagrammatic descriptions of a single service are consistent by construction, there is no need to check the consistency of a single service. However, in case of more complex formalisms, the same approach for combinations can be used for a single service as well.
Since a combination of services is the conjunction of the service specifications of a configuration, services are considered inconsistent if their combined specification equals the trivial specification *False*. However, checking for *False* is not sufficient since we require input-closed behavior. Thus, services exhibit an interaction problem if they have a trivial or non-input closed relation. Checking for input-enabledness requires the calculation of all reachable states of the transition. With the set of reachable states used to mark those trace positions identified by s, the equivalent of Equation 1 in terms of the μ calculus using the combined transition relation R is

$$\forall s.Reach(s) \Rightarrow \forall i_1,\ldots i_m.\exists o_1,\ldots o_n,t.\ R(s,i_1,\ldots i_m,o_1,\ldots o_n,t)$$

where *Reach* denotes the set of reachable states defined by

$$\mu\, Reach(s) \equiv Init(s) \vee \exists i_1,\ldots i_m,o_1,\ldots o_n,t.\ Reach(t) \wedge R(t,i_1,\ldots i_m,o_1,\ldots o_n,s)$$

with μ denoting the least fixed point used as interpretation of this recursive definition and *Init* the set of initial states.

If services are described using shared variables, a possible source of interaction is the assignment of values to variables. If two services assign different values to a shared variable, there is no interpretation for this assignment. Similar observations hold for communication actions, modeled by values assigned to channels. Note that feature interaction may as well occur combining two instantiations of the same feature (in case of the POTS specification, e.g., the service behaves differently if the initiator is being called prior to service start) as with the combination of two different features like call forwarding and call blocking.

2.6 Detecting Underspecification

In Section 2.3 we showed how the formalization of a service makes explicit the part where no behavior is defined by the description of the service. As stated above, formally, there is no difference between a single service and a combination of services. Thus, in the following, we will not make a distinction between those cases.

Similar to a nondeterministic automaton, a system of services is incomplete if either its initial states are incomplete or its transition relation. Using the above formalization, in the first case we have

$$\exists s,v.Init(s,\Box) \Rightarrow Init(\Box,v)$$

stating that there is either an undefined control or data state. In the second case, we have to check whether there is a reachable state with an outgoing transition which

- is no defined for all possible inputs
- does not completely define all outputs for a given input
- does not completely define the successor (control or data) state for a given input

More formally, incompleteness of a transition relation T is defined as

$$\lambda s, v.Reach(s,v) \Rightarrow \lambda i_1,\ldots i_m,o_1,\ldots o_n,v'.t.$$
$$T(s,v,i_1,\ldots,\bot,\ldots,i_m,o_1,\ldots,o_n,v',t) \wedge$$
$$T(s,v,i_1,\ldots,i_m,o_1,\ldots,\bot,\ldots,o_n,v',t) \wedge$$
$$T(s,v,i_1,\ldots,i_m,o_1,\ldots,o_n,\bot,t) \wedge$$
$$T(s,v,i_1,i_m,o_1,\ldots,o_n,v',\bot)$$

where *Reach* is defined as above.
Note that this definition does not exclude nondeterministic behavior if the nondeterministic behavior is explicitly stated by the service specification. In early phases nondetermism is often introduced into service specifications by abstracting from internal aspects of a system: an ATM may seemingly nondeterministically provide a customer with cash if abstracting from the current balance of the account. Therefore, from a methodical point of view it is necessary to support both nondeterminsm and underspecification but to distinguish between them.

As mention in Section 2.3, an incomplete service specification can be transformed into a (complete) system behavior by removing all $\bot$-traces from the set of traces of events of the service. This canonical transformation ca also be carried out on the level of the transition relation by simply excluding all $\bot$-transitions leading to a highly nondeterministic transition relation. To support a more operational interpretation for partial behavior, as, e.g., used in [HS01], undefined output is substituted by *nil* modeling that no signal is send; undefined successor control or data states are defined to remain unchanged. Again, such a canonical transformation can be easily defined on the level of the transition relation.

2.7 Automatic Support

The steps described above, translating diagrams, combining them and checking for interaction, can be performed automatically. The first step requires the translation into μ calculus. Additional information (system interface description, used variables) is needed for a translation as described in [HSE97]. Additionally, the description of a system state is generated from the description of the interface as well as the state information used by the features. More indirectly, the μ calculus-translator of AutoFocus/Quest [BL+00] can be used to translate the automaton representation of diagrams described by AutoFocus STDs to its μ calculus form. The μ calculus form of the state description can be generated from SSD descriptions used by AutoFocus to describe the interface and the data state of a system or component.

To combine the relations obtained from the diagrams configurations are needed to describe the intended activation of features. This combination is simply formed by conjunction of the service transition relation instantiated with the necessary parameters (Screen List, e.g.) and applied to the corresponding elements of the system state. This conjunction defines the transition relation for the system of the given configuration after applying the operational restriction described in Section 2.3. The operational restriction can be generated schematically from the interface and data state description similar to the system state description.

Since this example uses simple signals the system has a finite state space accessible to symbolic verification. Using a schematically generated variable order OBDDs representing

the transition relations can be kept sufficiently small to allow the generation of the set of reachable states. The check for an interaction problem is carried out using the relational μ calculus symbolic model checker μcke ([Bie97]). Here, all reachable states of the system transition starting from the initial state are checked for input actions missing an appropriate transition. Again, the μ calculus term can be generated schematically from interface and data space of the system. If such a missing transition is found, the counterexample component of μcke is used to generate execution traces leading to system states with conflicting feature requirements including the input actions which have no defined output actions.

Combining TCSC and CFBL as described by Table 1 including the grayed-out entries of party A we obtain a simple four-step execution trace leading to the first feature interaction problem mentioned in Section 1.2. μcke generates a counterexample given by assignments for the variables and channels of the system for each execution step starting from the initial state of the system and ending with the input action generating the conflict. μcke generates a μ calculus tableau representation. For a transparent use of this formalization, a different representation of the counterexample is needed, for example, in a MSC-like form. Figure 2 shows such a visualization of the counterexample for the TCSC/CFBL conflict.

3 Discussion of Results

In this article we introduced a formalization of the notions of a service, the combination of services, service (or feature) interaction and incompleteness, based on a simple model of system behaviors. To judge the usability of the presented formalization, we have to check whether it fulfills the requirements stated in Section 1.1:

It interprets diagrams independently, assigning a relation to each. Each one uses private and shared variables and part of the system interface offering the possibility to interact.

Service combination is simply the conjunction of the corresponding relations. Besides the system interface and the use of shared variables no further information is needed.

A feature-interaction problem is an inconsistent combination, checked by finding a reachable state with no defined transition for a certain combination of communication events.

Incompleteness of a service or a combination of services is an undefined behavior in a reachable state for a given input from the environment to the system.

Furthermore, inconsistency can be detected automatically including the generation of an example for such an inconsistency. Furthermore, incompleteness can be detected automatically as well. Our approach was applied to a benchmark specification used in the first Feature Interaction Detection Contest of the 5th international workshop on feature interactions, leading to comparable results as other approaches applied to it (see http://www-db.research.bell-labs.com/user/nancyg). However, besides being more complex, those approaches still have drawbacks compared to proposed approach: they either add implementation details, need a watch-dog process or different kinds of interaction detection, or detect superfluous “inconsistencies”. On the whole, the relational approach suggested here incorporates most of the advantages of the approaches discussed above while avoiding the above-mentioned disadvantages of a scenario-oriented form of description. Furthermore, the issue of the incompleteness of combined service descriptions is treated here, which is generally not considered in the other approaches.

As mentioned in Subsection 2.6, the first form of feature interaction of Subsection 1.2 is identified by our approach. However, the second was not identified due to the strict interpretation of the Chisel diagram of TCSC: since TCSC is defined as extension of POTS, TCSC becomes only active if the *direct* callee has the caller on the Screen List. [Hal98] argues that another important form, including the second example, are those interactions that correspond to invariants holding for the single features but not their combination. However, those additional invariants have to be stated explicitly using MSCs or temporal logic.[7], Using such an explicit formalization of additional invariants checking for their validity can be easily integrated in the presented framework. To deal only with Chisel diagrams, we can however use refinement, checking whether the combination of features refines the individual features, limiting the observation to the concerned actions. Methodically, refinement-checking as in [HSE98] can be used to check for a violation of a refinement between two scenarios.
Of course, this approach carries over to other formalisms (like WS1S) and their model checkers (like Mona [HJ+96]). Furthermore, service-based development of system requirements is not restricted to telecom systems. The increasing complexity of the functionality of embedded systems has lead to an increasingly service-oriented requirement specification in the automotive and aviation sector. Finally, the approach is not restricted to Chisel. Similar scenario-based description techniques, like (high level) MSCs or UML sequence diagrams, can be treated in the same manner supporting a modular, service-based development of requirements.

4 Bibliography

[AG+98] Aho, A. Gallagher, N. Griffeth, N. Schell, C. Swayne D. SCF3/Sculptor with Chisel. In: Kimbler, K. et al. (eds.) Proc. 5th Feature Interactions in Telecommunications and Software Systems. IOS Press, 1998.

[Bie97] Biere, A. Effiziente Modellprüfung des μ-Kalküls mit binären Entscheidungsdiagrammen. Ph.D. Thesis. Universität Karlsruhe, 1997.

[BL+00] Braun, P. Lötzbeyer, H. Schätz, B. Slotosch, O. Consistent Integration of Formal Methods. In: Graf, S. Schwartzbach, M. (eds.). Proc. TACAS'00. LNCS Vol. 1785. Springer, 2000.

[dB99] de Bousquet, L. Feature Interaction Detection Using Testing and Model Checking. In: Wing, G. Woodcock, J. Davies, J. (eds.) Proc. FM'99. LNCS 1708. Springer, 1999.

[GB+99] Griffeth, N. et al. Feature Interaction Detection Contest. Instructions. http://www-db.research.bell-labs.com/user/nancyg/ instructions.ps, 1999.

[Hal98] Hall, R. Feature Combination and Interaction Detection via Foreground/background Models. In: Kimbler, K. et al. (eds.) Proc. 5th Feature Interactions in Telecommunications and Software Systems. IOS Press, 1998.

[Hoa85] Hoare, C. Communication Sequential Processes. Prentice Hall, 1985.

[HSE97] Huber, F. Schätz, B. Einert, G. Consistent Graphical Specification of Distributed Systems. In: Fitzgerald, J. et al. (eds.) Proceedings of FME'97. LNCS Vol. 1313. Springer, 1997.

[JH+96] Henriksen, J. Jensen, J. Jørgensen, M. Klarlund, N. Paige, B. Rauhe, T. Sandholm, A. Mona: Monadic Second-Order Logic in Practice. In: Proc. Tools and Algorithms for the Construction and Analysis of Systems. LNCS Vol. 1019. Springer, 1996.

[7] See [SH99] and [Bie97] for details on the μ-formalization of MSCs and temporal logic expression.

[JZ00] Jackson, M. Zave, P. New Feature Interactions in Mobile and Multimedia Telecommunication Services. In: Calder, M. et al. (eds.) Proc. 6th Feature Interactions in Telecommunications and Software Systems. IOS Press, 2000.

[Krü00] Krüger, I. Distributed System Design with Message Sequence Charts, Dissertation, Technische Universität München, 2000.

[Lam93] Lamport, L. Specification and Verification of Concurrent Programs. In: de Bakker, J.W. et al. (eds.). A Decade of Concurrency. LNCS Vol. 803. Springer, 1993.

[LT89] Lynch, N. Tuttle, M. An Introdcution to Input/Output Automata. CWI Quaterly 3(2). 1989.

[Par76] Park, D. Finiteness is μ-Ineffable. Theoretical Computer Science 1(3). 1976.

[SH99] Schätz, B. Huber, F. Integrating Formal Description Techniques. In: Wing, G. et.al. (eds.) Proc. FM'99. LNCS 1709. Springer, 1999.

[Tur00] Turner, K. Formalizing the Chisel Feature Notation. In: Calder, M. et al. (eds.) Proc. 6th Feature Interactions in Telecommunications and Software Systems. IOS Press, 2000.

[Zav99] Zave, P. FAQ Sheet on Feature Interaction. AT&T 1999. http://www.research.att.com/~pamela/faq.html

5 Appendix

As section 2.2 only describes the mapping of Chisel diagrams using an example, we now give a formal description of the mapping to the system model. Instead of giving a direct mapping, we will follow the outline of section 2.2 and transform Chisel diagrams into an intermediate form of automata-based description called STD. For this STD form, a translation into a relational description is used and possible execution sequences and system behavior is defined along the lines of [LT89].

Definition 1. *A system interface consists of families of unique input and output channel names*

$$i_1,\ldots,i_m,\quad o_1,\ldots,o_n$$

and their corresponding types (sets of values)

$$I_1,\ldots,I_m,\quad O_1,\ldots,O_n$$

Definition 2. *The behavior of a system with a system interface as described above containing input channel types $I_1,\ldots,I_m$ and output channels types $O_1,\ldots O_n$ is defined as an input closed set T of traces of events with*

$$T \subseteq (I_1 \cup \ldots \cup I_m \cup O_1 \cup \ldots \cup O_n)^{\Box} \text{ and}$$

$$\forall r,s.\ r \circ s \in T.\ \forall i_1,\ldots i_m.\ \forall o_1,\ldots o_n,t.\ r \circ (i_1,\ldots i_m,o_1,\ldots o_n) \circ t \in T$$

The System model of a system with input channel types $I_1,\ldots I_m$ and output channels types $O_1,\ldots O_n$ is defined as set of all behaviors of such a system.

Definition 3. *Given a system interface containing input channel types $I_1,\ldots,I_m$ and output channels types $O_1,\ldots O_n$, a service of this system is an input-closed partial set S of traces of events with*

$$S \subseteq (I^{\Box}_1 \cup \ldots \cup I^{\Box}_m \cup O^{\Box}_1 \cup \ldots \cup O^{\Box}_n)^{\Box} \text{ and}$$

$$\forall r,s.\ r \circ s \in S.\ \forall i_1,\ldots i_m.\ \forall o_1,\ldots o_n,t.\ r \circ (i_1,\ldots i_m,o_1,\ldots o_n) \circ t \in S$$

As mentioned in Section 2.1, in Chisel diagrams actions are not assigned to channels but use addresses. However, since those actions can be trivially mapped to actions of the system model as sketched in Section 2.1, we will assume this representation in the following definitions.

A Chisel diagram can be interpreted as a bipartite tree $(I, A, T, V, r, m, a, n, b)$. Nodes from $I \cup A$ are marked via the function m with one input action (actions of the environment of the system), and output actions of the system (or a sequence of those), res. The root of the tree r is marked with an input action. Additionally, all nodes are marked with a positive integer via

the function n. The edges of the tree T are a subset of $I \times A \cup A \times I$. Finally, edges of the tree can be marked with Boolean condition via the function b.

Definition 4. *A Chisel diagram of a service of a system with an interface as described above and variables $v_1,\ldots v_k$ of type $V_1,\ldots V_k$ is a bipartite tree $(I, A, T, V, r, m, a, n, b)$ with disjunct set of nodes I and A, a family $V = (V_i)$ of variable types, a set T of edges of the tree, i.e. a subset of $I \times A \cup A \times I$, a root $r \in I$, and four labeling functions*

$$m : I \to \bigcup_j \{i_j.w \mid w \in I_j\}$$
$$m : A \to (\bigcup_j \{o_j.w \mid w \in O_j\})^+$$
$$a : A \to (\bigcup_j \{v_j = w \mid w \in V_j\})^*$$
$$n : A \to \mathbb{N}^+$$
$$b : I \times A \to (\bigcup_j \{v_j = w \mid w \in V_j\})^*$$

The following definition describes the transformation of a Chisel diagram into a STD as defined in [HSE97]. A STD (state transition diagram) corresponds to graph representing an extended finite automaton with input and output events.

Definition 5. *Given a Chisel diagram $(I, A, T, V, r, m, a, n, b)$ as defined above, the corresponding STD is defined as*

Set of states labeled with natural numbers: $\{s \mid \forall a \in A.\, s = n(a)\} \cup \{0\}$

Initial state: 0

Transition relation R defined by

$$\{(0 \xrightarrow{b(r,o):m(r)/m(o):a(o)} n(o)) \mid (r,o) \in T\} \cup$$
$$\{(n(p) \xrightarrow{b(i,o):m(i)/m(o):a(o)} n(o)) \mid (p,i) \in T \wedge (i,o) \in T \wedge \forall j.(o,j) \in T\} \cup$$
$$\{(n(p) \xrightarrow{b(i,o):m(i)/m(o):a(o)} 0) \mid (p,i) \in T \wedge (i,o) \in T \wedge \forall j.(o,j) \notin T\}$$

Semantically, a STD basically corresponds to an extended finite automaton $(S, V_1 \times \ldots \times V_k, I_1 \times \ldots \times I_m, O_1 \times \ldots \times O_n, T, s, s_1 \times \ldots \times s_k)$ defined by a set of states S, a set of possible variable assignments $V_1 \times \ldots \times V_k$, a set of input and output events $I_1 \times \ldots \times I_m$ and $O_1 \times \ldots \times O_n$, res., a transition relation

$T \subseteq S \times (V_1 \times \ldots \times V_k) \times (I_1 \times \ldots \times I_m) \times (O_1 \times \ldots \times O_m) \times (V_1 \times \ldots \times V_k) \times S$

and a set of possible initial states and variable assignments $s, s_1 \times \ldots \times s_k$. Following the usual convention, we use the short-hand notation

$$(s,v) \xrightarrow{(i,o)} (t,w) \quad \equiv \quad (s,v,i,o,w,t) \in T$$

[HSE98] describes a translation of a STD into the relational μ calculus and defines the construction of the transition relation and the relation characterizing the initial state. For the formalization of Chisel we basically use the same translation, however, skipping a nil-assignment for undefined input or output actions. This is due to the fact that here we are interested in a loose interpretation of the STD (supporting underspecification) rather than an interpretation close to programming languages. Thus, in the relational approach a system with interface $(i_1,\ldots,i_m),(o_1,\ldots,o_n)$ of types $(I_1,\ldots,I_m),(O_1,\ldots,O_n)$ and variables $(v_1,\ldots v_k)$ of types $(V_1,\ldots V_k)$ is described by two relations with the arities

- $T(s\ S, v_1\ V_1, \ldots v_k\ V_k, i_1\ I_1, \ldots, i_m\ I_m, o_1\ O_1, \ldots, o_n\ O_m, v_1' V_1, \ldots, v_k' V_k, t\ S)$
- $Init(s\ S, v_1\ V_1, \ldots v_k\ V_k)$

to describe the transition relation and the relation characterizing the initial states.

Definition 6. *The behavior of an STD (and thus of a system specified by Chisel diagrams) is defined as the admissable closure of the set of finite sequences of input/output event pairs* $(i,o)_0 \circ (i,o)_1 \circ (i,o)_2 \circ \ldots \circ (i,o)_n$ *from execution sequences (sequences of states/variables and input/output events) with*

$$(s,v)_0 \circ (i,o)_0 \circ (s,v)_1 \circ (i,o)_1 \circ (s,v)_2 \circ (i,o)_2 \circ \ldots \circ (i,o)_n \circ (s,v)_{n+1}$$

such that

$$(\forall i.\ (s,v)_i \xrightarrow{(i,o)_i} (s,v)_{i+1}) \Rightarrow (s,v)_0 \Rightarrow s \Rightarrow s_1 \Rightarrow \ldots \Rightarrow s_k$$

Definition 7. *Given system descriptions of a system A with an interface* $(i_1,\ldots,i_m)$ *and* $(o_1,\ldots,o_n)$ *of types* $(I_1,\ldots,I_m)$ *and* $(O_1,\ldots,O_n)$ *and variables* $(v_1,\ldots,v_k)$ *of types* $(V_1,\ldots,V_k)$ *and a system B with an* $(i_1',\ldots,i_q')$ *and* $(o_1',\ldots,o_r')$ *of types* $(I_1',\ldots,I_p')$ *and* $(O_1',\ldots,O_q')$ *and variables* $(w_1,\ldots,w_p)$ *of types* $(V_1',\ldots,V_p')$*, such that*

$$\{(i_1,I_1),\ldots,(i_1,I_1)\} \subseteq \{(i_1',I_1'),\ldots,(i_q',I_q')\}$$

$$\{(o_1,O_1),\ldots,(o_1,O_1)\} \subseteq \{(o_1',O_1'),\ldots,(o_r',O_r')\}$$

$$\{(v_1,V_1),\ldots,(v_1,V_1)\} \subseteq \{(w_1,V_1'),\ldots,(w_p,V_p')\}$$

system A is extended to the interface and variables of system B by defining the new transition relation T_A' *and relation* $Init_A'$ *characterizing the initial state on basis of the original relations* T_A *and* $Init_A$ *as*

$$T_A'(s\ S, w_1\ V_1', \ldots, w_p\ V_p', i_1\ I_1', \ldots, i_q'\ I_q', o_1\ O_1', \ldots, o_r'\ O_r', w_1'\ V_1', \ldots, w_p'\ V_p', t\ S) =$$
$$T_A(s, v_1, \ldots v_k, i_1, \ldots, i_m, o_1, \ldots, o_n, v_1', \ldots, v_k', t)$$

$$Init_A'(s\ S, w_1\ V_1' \ldots w_p\ V_p') = Init_A(s, v_1, \ldots v_k)$$

Since now two system descriptions of services with different interfaces and variables can be both extended to their least common superset of interface and variables, arbitrary services can be combined into one system. The transition relation and the relation characterizing the initial state are simply defined as the conjunction of the corresponding relations of the individual systems; the control state of the relations is the product of the individual control states of the original relations. On basis of this definition, the interaction problem for combinations of services can be defined.

Definition 8. *A system of services with a transition relation T and a relation Init characterizing its initial states as described above is defined to exhibit a feature-interaction problem if either*

$$Init(s,v) = False$$

or

$$\neg(\exists s,v.Reach(s,v) \wedge \forall i_1,\ldots i_m.\exists o_1,\ldots o_n,v'.t.T(s,v,i_1,\ldots i_m,o_1,\ldots o_n,v',t))$$

where Reach *denotes the set of reachable states defined by*

$$\mu\, Reach(s,v) \equiv Init(s,v) \vee \exists i_1,\ldots i_m,o_1,\ldots o_n,v',t.\ Reach(t) \wedge T(t,v',i_1,\ldots i_m,o_1,\ldots o_n,v,s)$$

with μ *denoting the least fixed point used as interpretation of this recursive definition.*

Finally, using the same formalization we can formalize the notion of an incomplete feature specification.

Definition 9. *A system of services with a transition relation T and a relation Init characterizing its initial states as described above is defined to be an incomplete feature specification if either*

$$\exists s,v.Init(s,\bot) \wedge Init(\bot,v)$$

or

$$\begin{aligned}&\exists s,v.Reach(s,v) \wedge \forall i_1,\ldots i_m,o_1,\ldots o_n,v'.t.\\ &\quad T(s,v,i_1,\ldots,\bot,\ldots,i_m,o_1,\ldots,o_n,v',t) \wedge\\ &\quad T(s,v,i_1,\ldots,i_m,o_1,\ldots,\bot,\ldots,o_n,v',t) \wedge\\ &\quad T(s,v,i_1,\ldots,i_m,o_1,\ldots,o_n,\bot,t) \wedge\\ &\quad T(s,v,i_1,i_m,o_1,\ldots,o_n,v',\bot)\end{aligned}$$

where Reach *is defined as above.*

Note that this definition does not exclude nondeterministic behavior if the definition is explicitly stated by the feature specification.