An Abstract Component Model

Egon Börger

Dipartimento di Informatica, Universita di Pisa
http://www.di.unipi.it/~boerger

For details see Chapter 3.1 of:

E. Börger, R. Stärk

Abstract State Machines

A Method for High-Level System Design and Analysis

Springer-Verlag 2003

For update info see AsmBook web page:

http://www.di.unipi.it/AsmBook

Goal of abstract framework for component composition

- Provide concepts for components which
  - come with interface specifications depending on views
  - export and import parameterized services under usage constraints
  - can be composed and refined by connectors
- based upon abstract notions of
  - specification
  - view
  - service
  - constraint (model of logic)
  - connector

Component Structures

- COMPONENT: domain of abstract components equipped with appropriate abstract functions:
  - Exports: COMPONENT $\rightarrow$ Powerset(EXPSERVICE)
    yielding the services exported by a component
  - Imports: COMPONENT $\rightarrow$ Powerset(IMPSERVICE)
    yielding the services imported by a component
  - SERVICE = EXPSERVICE $\cup$ IMPSERVICE
    - ServiceParam: SERVICE $\rightarrow$TYPE $\times$ MODE yielding the parameters of a service

      ServiceResType: SERVICE $\rightarrow$TYPE yielding the "result" type
    - MODE= {in,out,inout}
    - Component: SERVICE $\rightarrow$ COMPONENT yielding the component to which a service belongs

Abstract constraints on usage of component services

- Constraint: COMPONENT $\rightarrow$ CONSTRAINT

  expressing allowed usages (e.g.application ordering) of the services which are exported by a component

- ImportStructure: EXPSERVICE $\rightarrow$ USESTRUCTURE

  expressing allowed usages of an exported service (in relation to other services imported - in the body of the exporting component - and possibly used to execute the exported service), to be met by the constraints of the components from where those imported services are imported

  – ContainedServices: USESTRUCTURE $\rightarrow$ Powerset(IMPSERVICE)

  yielding the set of imported services which are contained in a given use structure

  – MeetsConstraint : USESTRUCTURE $\times$ CONSTRAINT $\rightarrow$ BOOL

  indicating whether the usage of imported services (in the body of a component) is consistent with the given constraints (typically of the components which export those services)

Component specifications and related functions

- ProvidedSpec : VIEW $\times$ EXPSERVICE $\rightarrow$ SPEC
  RequiredSpec : VIEW $\times$ IMPSERVICE $\rightarrow$ SPEC

  yielding the specification of a service under a given view. The class SPEC of allowed interface descriptions is deliberately kept abstract.
- SatisfiesSpec : VIEW $\times$ SPEC $\times$ SPEC $\rightarrow$ BOOL
  indicating whether a given (expservice) specification "satisfies" a given (impservice) specification
  - so that the service to be exported can safely be "plugged" into the component where to import it

  The satisfaction relation is deliberately kept abstract, it depends on the underlying class SPEC and on the logic adopted to describe the intended semantical relations between specifications.

Connectors and well-formedness assumptions

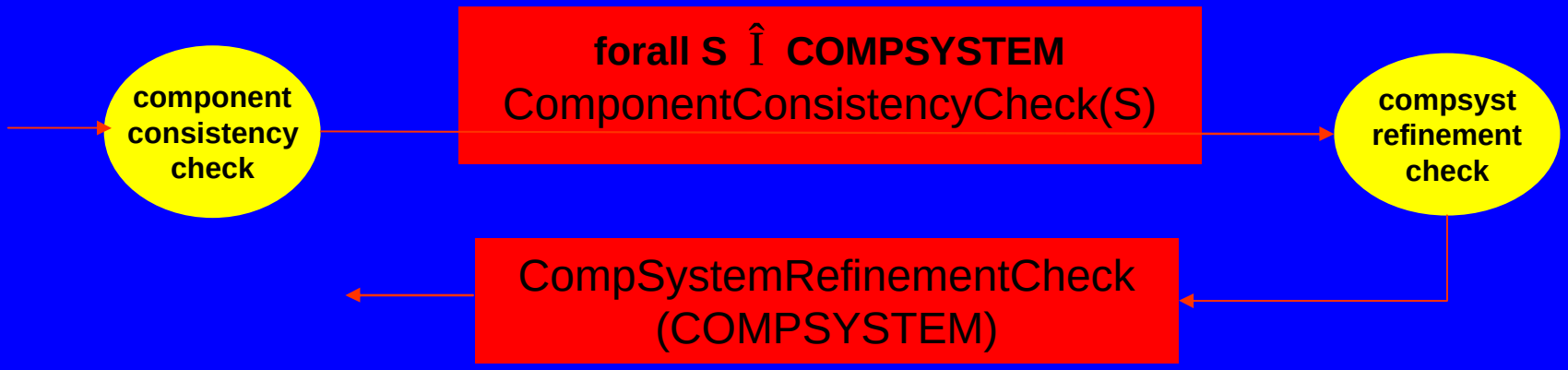
- CONNECTOR: domain of connectors connecting required (imported) services to exported services
  - Connector: IMPSERVICE $\times$ EXPSERVICE $\rightarrow$ CONNECTOR mapping the service required by a component to an exported service of another component, with two projection functions:
    - ImportService: CONNECTOR $\rightarrow$ IMPSERVICE
    - ExportService: CONNECTOR $\rightarrow$ EXPSERVICE
  - AbstractCONNECTOR $\subseteq$ CONNECTOR with function AbstractConnector. Abstract connectors are allowed not to obey the consistency conditions checked in the connector model below.
- Axiom: Every import service is connected to at most one exported service by either a connector or an abstract connector
- Axiom: No export service uses in its body an import service to which it is linked by a connector-chain (non-cyclic connectors)

Component Systems

- Component Systems are sets of component structures S as described above which satisfy the above stated well-formedness assumptions
  - in a component system COMPSYSTEM, the sets and functions of each of its component structures S are parameterized by S, although notationally we suppress the parameter when it is clear from the context
- Well-formedness component system assumption (axiom):
  - Every component/connector in a component system belongs to exactly one component structure of the system

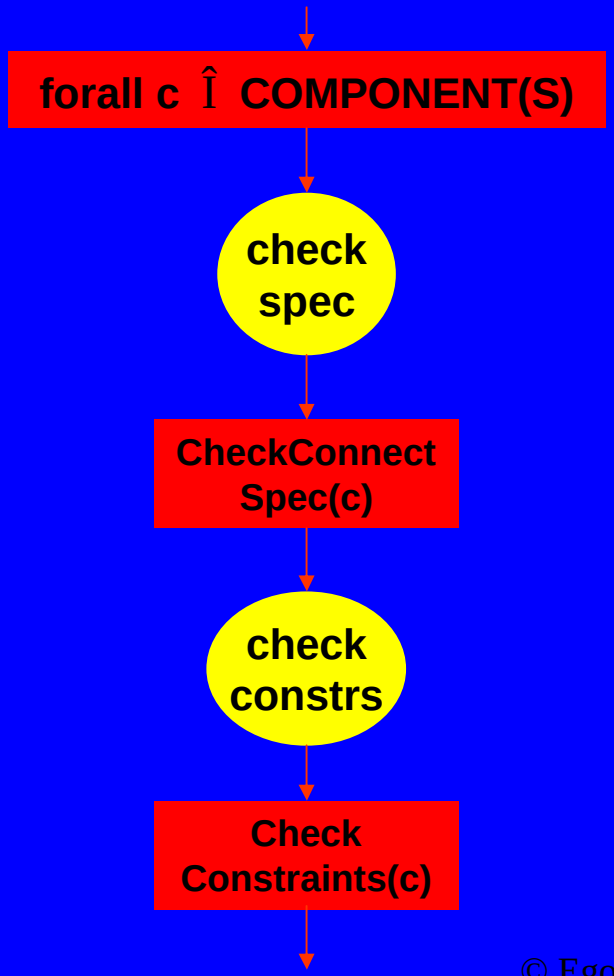
Two-fold Goal: a consistency check for component structures and for component system refinements

- Define a consistency notion for
  - component structures (correctness of connections and satisfaction of constraints)
  - refinement of component structures (correct refinement of types, views, components, connectors) belonging to component systems
- Specify a machine to check these two properties for a given component system with a proposed refinement relation
- The consistency checks should be implementable as stand-alone process
  - for each single component structure
  - for each refinement pair of two component structures of a component system



ComponentConsistencyCheck(S) Machine

checking for each component of a component structure S its connectors and constraints:
1) the connectors to connect import services to export services with matching signature and specification
2) the constraints of all used components (those from where services are imported to execute a service exported by the component) to meet the related use structure



CheckConnectSpec (c) Machine checking that **connectors connect every import service to an export service with matching signature & spec (for every view)**

≅ denotes equality of type and mode

forall $s_{imp} \in$ Imports(c)

If $\forall$s Connector(s_{imp} , s) = undef then

ImpServiceNotConnected(s_{imp})

Else let $s_{exp} = \iota s$(Connector(s_{imp} , s) is defined)

If not equivSignatures (s_{imp} , s_{exp})

then output error-in-connector (s_{imp} , s_{exp}, no signature match)

forall v$\in$ View: if SatisfiesSpec(v, ProvidedSpec(v,s_{exp}), RequiredSpec(v,s_{imp}) = false

then output error-in-connector (s_{imp} , s_{exp}, no spec match)

equivSignatures(s,s') ^{o}

|ServiceParam(s)| = |ServiceParam(s')| & " 1 ≤i ≤n

ith(**ServiceParam(s)**) ≅ ith(**ServiceParam(s')**)

where **n = |ServiceParam(s)|**

ImpServiceNotConnected(s) ^{o}

If for all s' $\in$ EXPSERVICE

AbstractConnector(s,s') = undef

then error:= impservice s has no connector

CheckConstraints (c) Machine

checking for each exported service the constraints of all used components (those from where services are imported to execute the exported service) to meet the related use structure

forall $s_{exp} \in$ Exports(c)

forall c' $\in$ UsedComponents(s_{exp})

If MeetsConstraint(ImportStructure(s_{exp}),Constraint(c')) = false

Then output error (s_{exp} , c', component constraint is violated)

UsedComponents(s) ^{o}

$\{c | \exists\ s_{imp} \in$ ContainedServices(ImportStructure(s)) and
$\exists s_{exp}$ Connector(s_{imp}, s_{exp}) is defined and
$s_{exp} \in$ Exports(c)$\}$

CompSystemRefinementCheck(COMPSYSTEM) Machine

forall R,S ∈ COMPSYSTEM such that R-isRefinedBy-S

checkType ViewComp

Check TypeViewComp Refinement(R,S)

check constrs

CheckConnector Refinement(R,S)

CheckTypeViewCompRefinement(R,S)

- **CheckTypeRefinement (R,S)**
  CheckViewRefinement (R,S)
  CheckComponentRefinement (R,S)

CheckType/ViewRefinement (R,S) Machine

checking that abstract types and views are subsets of the refined ones

CheckTypeRefinement (R,S) °

If not Types(R) ⊆ Types(S)

then set error-missing-types-in-refinement(R,S)

CheckViewRefinement(R,S) °

If not Views(R) ⊆ Views(S)

then set error-missing-views-in-refinement(R,S)

Machine for CheckComponentRefinement (R,S)

checking that

forall $c_R \in$ COMPONENT(R), $c_S \in$ COMPONENT(S) s.t.
ComponentName(c_R) = ComponentName(c_S)

If Imports(c_R) $\neq \emptyset$ then

If not $c_R \approx_{comp} c_S$ then output error-in-comp-refinement(c_R , c_S, a component with import services cannot be refined)

If not $c_R \approx_{exp} c_S$ then output error-in-comp-refinement(c_R , c_S, abstract & refined components must have identical expservices)

component-refined(R,S, c_R):=(S, c_S)

If for some $c_R \in$ COMPONENT(R) there is no $c_S \in$ COMPONENT(S)
s.t. ComponentName(c_R) = ComponentName(c_S)
then output error-in-comp-refinement (R,S, each component must have a refined version)

NB. Each c_R admits at most one c_S with same component name, by well-formedness and assumption on ComponentName: assumed as injective in COMPONENT(s) and with range disjoint from connector names

Defining Equivalence of Components

$c \approx_{exp} c'$ iff — equivalence wrt export services

- Constraint(c) ≈ Constraint(c')
- c,c' have the same number of export services
- export services of c,c' with same name are $\approx_{service}$

$c \approx_{comp} c'$ iff — equivalence wrt export/import services & wrt import structure

- $c \approx_{exp} c'$
- export services of c,c' with same name
  - have the same ImportStructure
- import services of c,c' with same name are $\approx_{service}$

Defining Service Equivalence

- $S \approx_{service} S'$ iff
  - s, s' have the same name
  - equivSignatures(s,s')
  - For each view v: Provided/RequiredSpec(v,s) $\approx_v$ Provided/RequiredSpec(v,s')

Refinement of connectors: an example

- Idea: For each connector cn in R, either cn $\in$ S or there is a component cni that refines the connector in S
- cn in R is refined by cni in S iff
  - cn and cni have the same name
  - cni exports a service that matches the import service of cn
  - cni imports a service that matches the export service of cn
  - Exl. connector C {A::s1 $\rightarrow$ [ ] $\rightarrow$ B::s2}
    - connects import service s2 of B to export service s1 of A
    - a valid refinement is
      - connector {A::s1 $\rightarrow$ C::s1}
      - connector {C::s2 $\rightarrow$ B::s2}

      with component C in S offering export service s2 to B and import service s1 to A

Machine for CheckConnectorRefinement (R,S)

forall $cn_R \in$ CONNECTOR(R), $cn_S \in$ CONNECTOR(S) s.t.

Name(ImportService(cn_R))= Name(ImportService(cn_S))

Component(ImportService(cn_R)) is comp-refined by Component(ImportService(cn_S))

If both or none of cn_R, cn_S are abstract ($\in$ AbstractCONNECTOR)

then if ExportService(cn_R) $\approx_{refine}$ ExportService(cn_S)

then connector-refinement-ok(cn_R, cn_S)

else CheckConnectorRefinement(cn_R, cn_S)

If cn_R is abstract and cn_S is not abstract

then CheckConnectorRefinement(cn_R, cn_S)

If cn_R is not abstract and cn_S is abstract

then output error-in-connector-refinement(cn_S, only abstract connectors refinable)

es(cn_R) $\approx_{refine}$ es(cn_S) iff both have same es-name & Component(es(cn_R)) is comp-refined by Component(es(cn_S))

c is comp-refined by c' iff component-refined(R,S,c)=(S,c')

Machine for CheckConnectorRefinement (cn_R, cn_S)

let e_R = ExportService(cn_R), E_R = Component(e_R),
i_S = ImportService(cn_S)

If there is no comp-refinement E_S of E_R exporting a service e_S with same name as e_R

then Output error-in-connector-refinement(cn_S, no service matching $E_R :: e_R$ in S)

elseif in a component of S there is

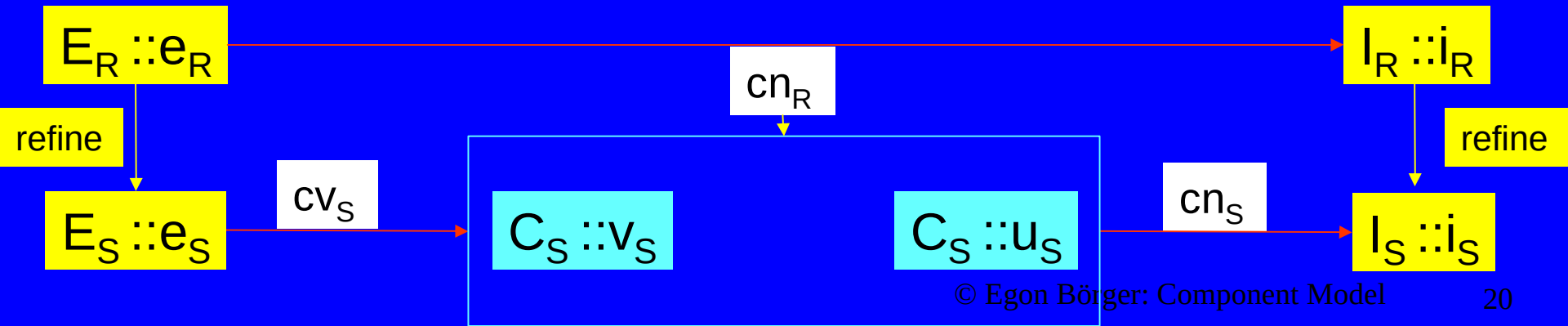
no import service v_S connected to an e_S as above

no export service u_S to which i_S is connected

then Output error-in-connector-refinement(cn_S, refining connector missing)

else connector-refinement-ok(cn_R , cn_S)

i connected to e iff Connector(i,e) or AbstractConnector(i,e) is defined



Exercise (on sequentialization)

- Show that the ComponentConsistencyCheck(S) ASM defined above is equivalent to the following 1-step machine:

  forall c ∈ COMPONENT(S)

  CheckConnectSpec(c) seq CheckConstraints(c)

  and that the CompSystemRefinementCheck(COMPSYSTEM) ASM defined above is equivalent to the following 1-step machine:

  forall R,S ∈ COMPSYSTEM such that R-isRefinedBy-S

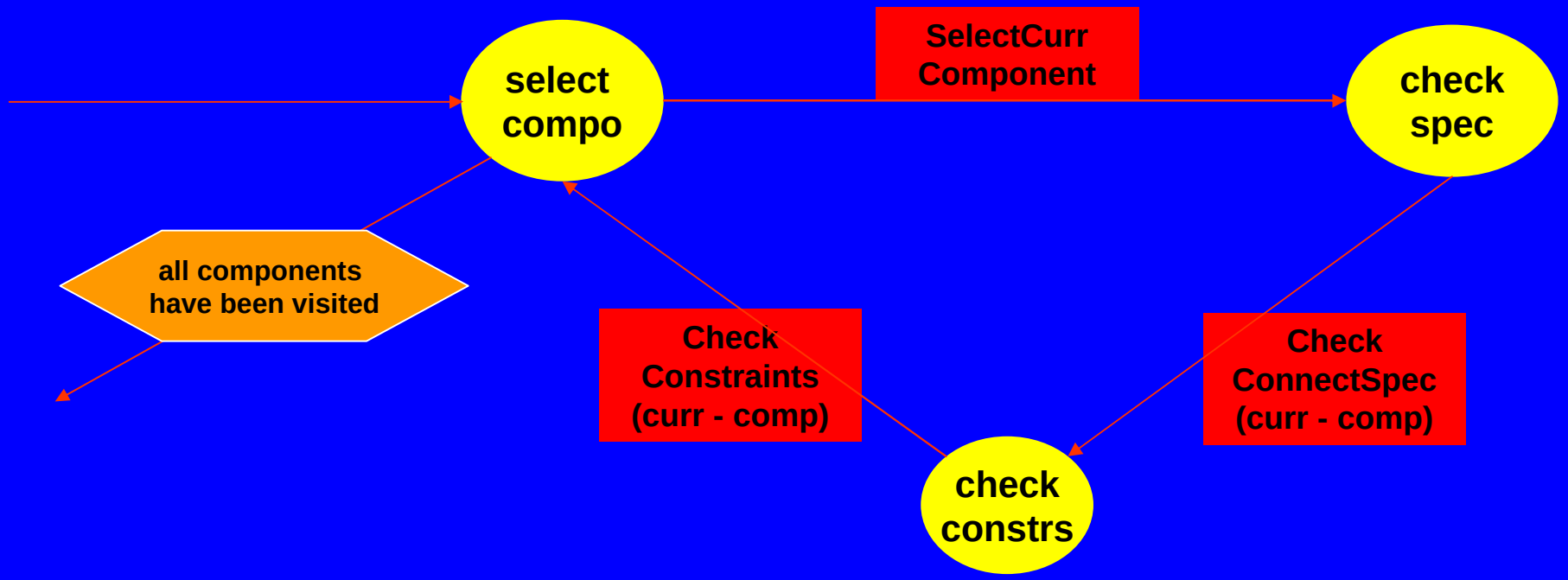
  CheckTypeViewCompRefinement(R,S)

  seq CheckConnectorRefinement(R,S)

  where seq denotes the ASM sequentialization operator defined in [BÖrger&Schmid2000].

Exercise (on refining forall-machines)

- Refine the machine ComponentConsistencyCheck(S) by an iterated control state ASM working in each iteration on one component only, but with an abstract scheduling for the iteration order. Prove the equivalence.
  - Hint (A. Sünbül, Dissertation, Fig. 5.3): use iterated choose as shown below.



SelectCurrComponent ° choose c Î COMPONENT(S) such that visited(c)=false
curr - comp:= c
visited(c):= true

Exercises

- Refine CompSystemRefinementCheck(COMPSYSTEM) by an iterated control state ASM which works in each iteration step on one refinement pair and prove the refinement correctness.
  - Hint: Define as in the preceding exercise a SelectCurrRefinementPair machine followed by CheckTypeViewCompRefinement and CheckConnectorRefinement (see A. Sünbül, Dissertation, pg. 89) .
- Refine the abstract 1-step machines CheckConnectSpec, CheckConstraints, CheckComponentRefinement, CheckConnectorRefinement to iterated machines working in each iteration on one import service, export service, component, connector respectively, but with an abstract scheduling for the iteration order. Show the correctness of the refinement.
  - For a solution which uses the choose-construct see A. Sünbül, Dissertation, Fig. 5.4, Fig.5.6, pg.91, pg.95.

Exercise

- Define ASMs to compute UsedComponents , the signature equivalence notion equivSignatures(s,s') , and the component and service equivalence notions c $\approx_{comp,exp}$ c', c $\approx_{comp}$ c', s $\approx_{service}$ s' specified above.
  - For an implementation using the choose-construct see A. Sünbül, Dissertation, Fig. 5.2, 5.1, 5.8, 5.7, 5.9.

Projects

- Use the above described approach for abstract definitions of components, connectors, interface constraints, etc. to provide a rigorous abstract model for real-life multi-platform middleware techniques for component composition (e.g. Corba) or for the component concept of a real-life component based programming system (e.g. JavaBeans) or of your favorite glue-code providing scripting language.

References

- A. Sünbül: Architectural Design of Evolutionary Software Systems in Continuous Software Engineering
  - Dissertation, Technical University of Berlin, 2001
- E. Börger, J. Schmid : Composition and Submachine Concepts for Sequential ASMs
  - Proc. CSL'2000, LNCS 1862, 2000
- E. Börger, R. Stärk: Abstract State Machines. A Method for High-Level System Design and Analysis Springer-Verlag 2003, see http://www.di.unipi.it/AsmBook
- R. Stärk, J. Schmid, E. Börger :Java and the Java Virtual Machine: Definition, Verification, Validation Springer-Verlag 2001 http://www.inf.ethz.ch/~jbook