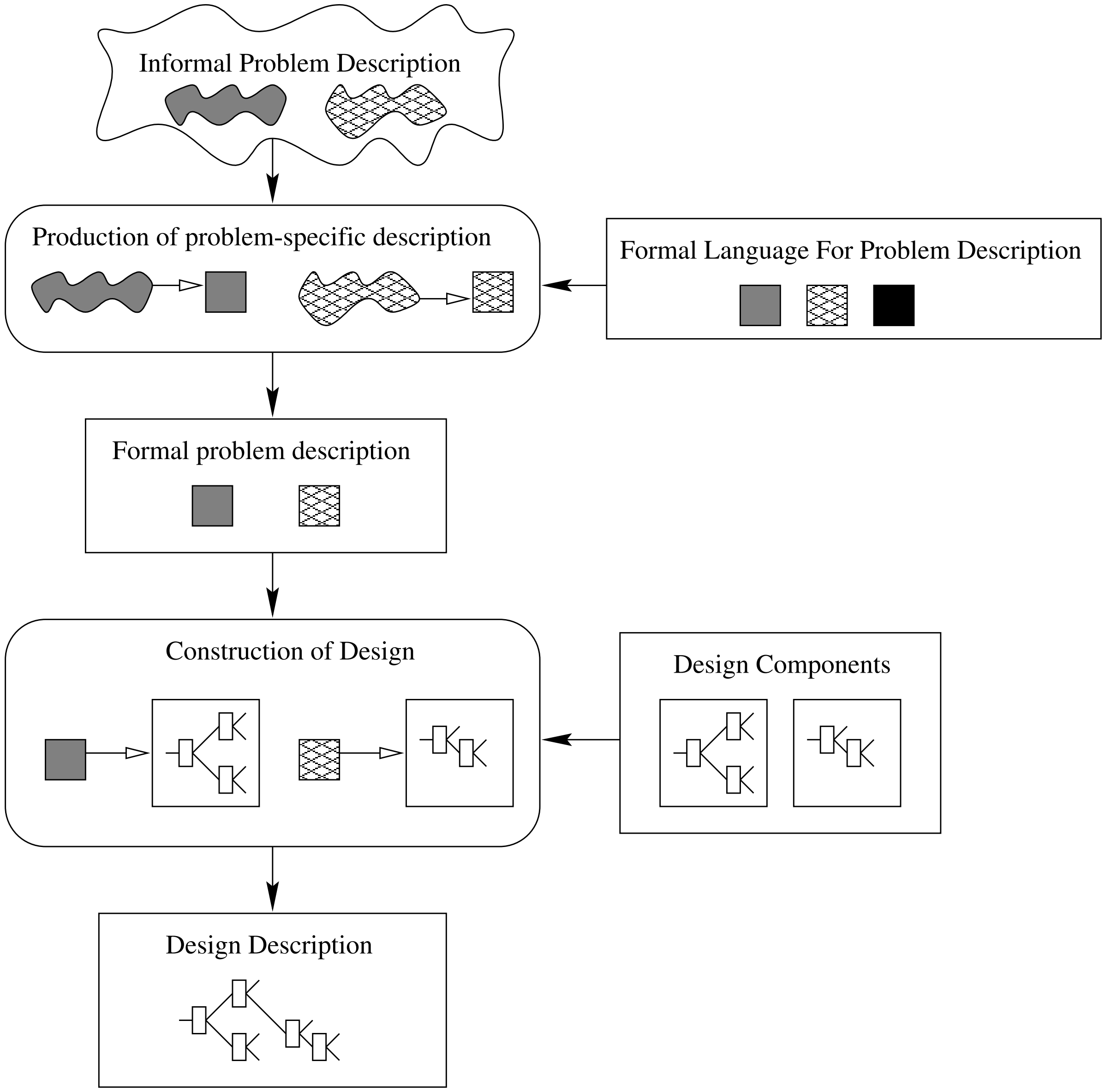
Automating Software Engineering

One way to reduce costs (and perhaps raise quality) in software engineering is to automate where we can. We look at examples of this in:

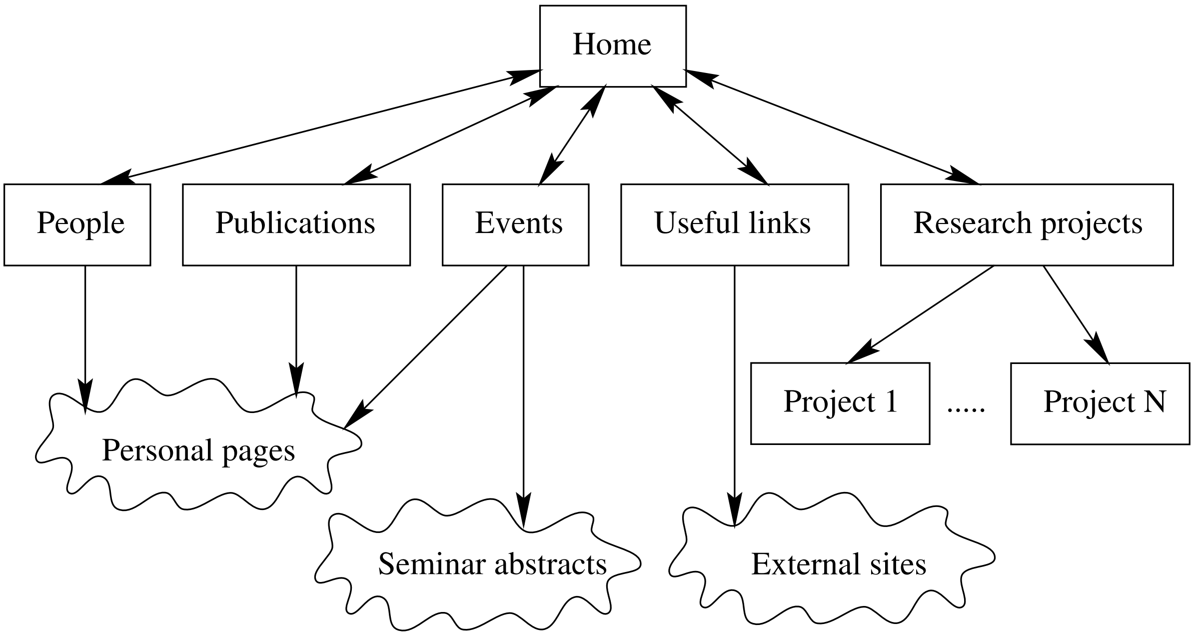
- Automating component-based design.
- Linking components to rationale.
- Automating traceability.
- Connecting to established lifecycles.
- Pragmatics of formal design.

Further reading in Robertson & Agusti 1999 *Software Blueprints*.

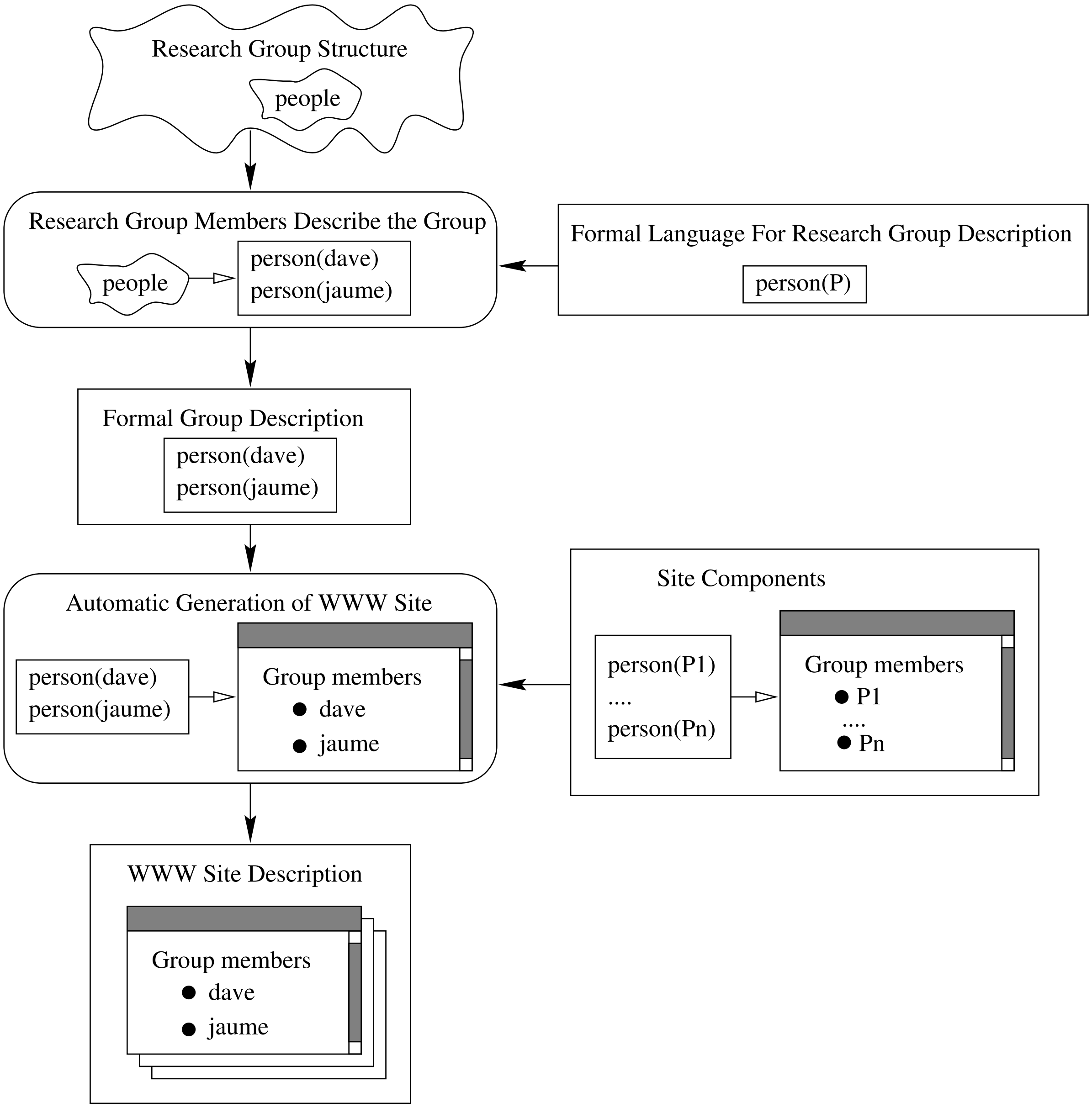
Automating Component-based Design



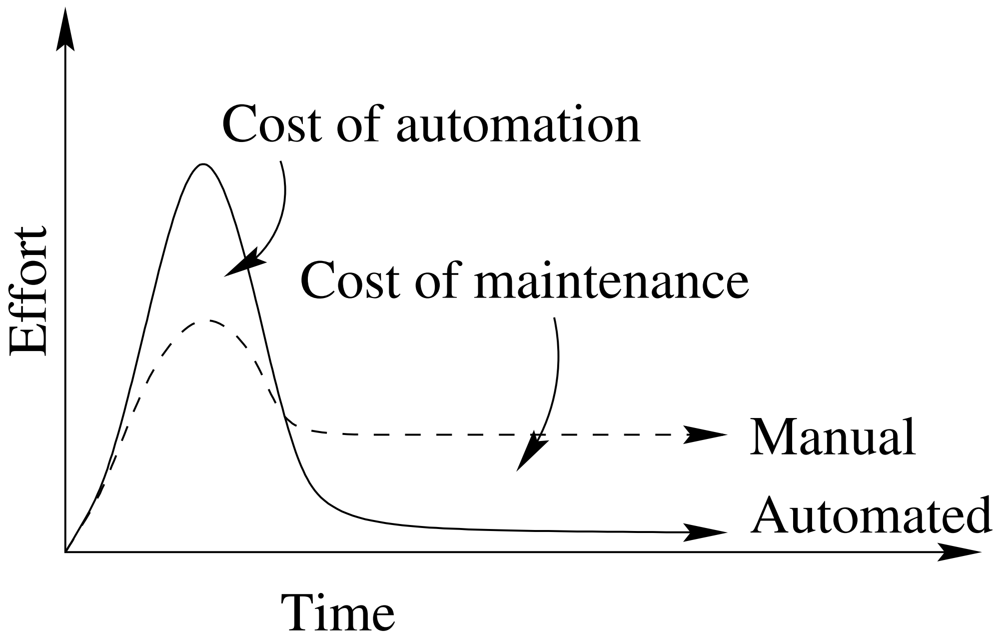
Research Group Web Site



Automating Web Site Generation

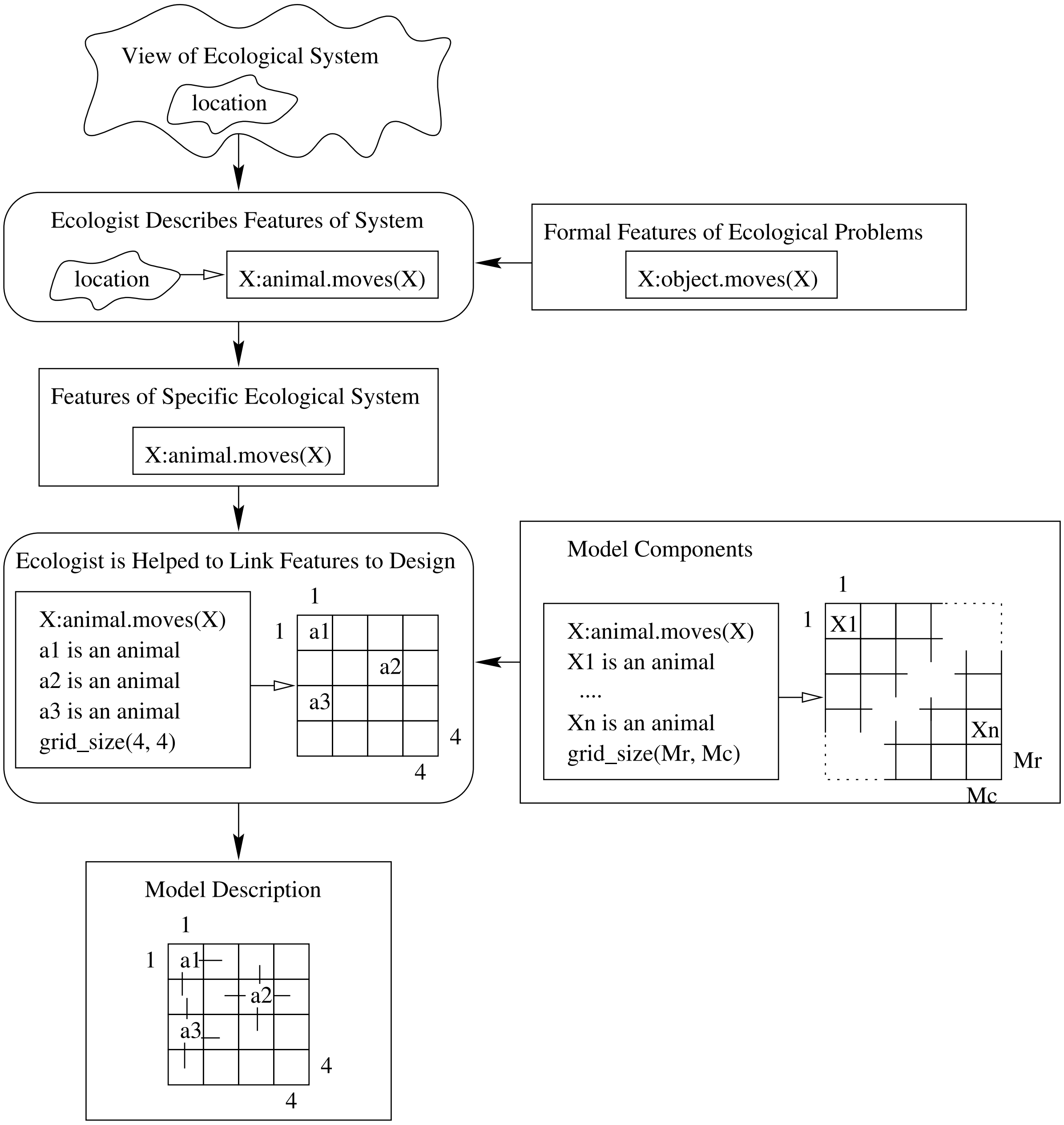


The Effect on Software Cost

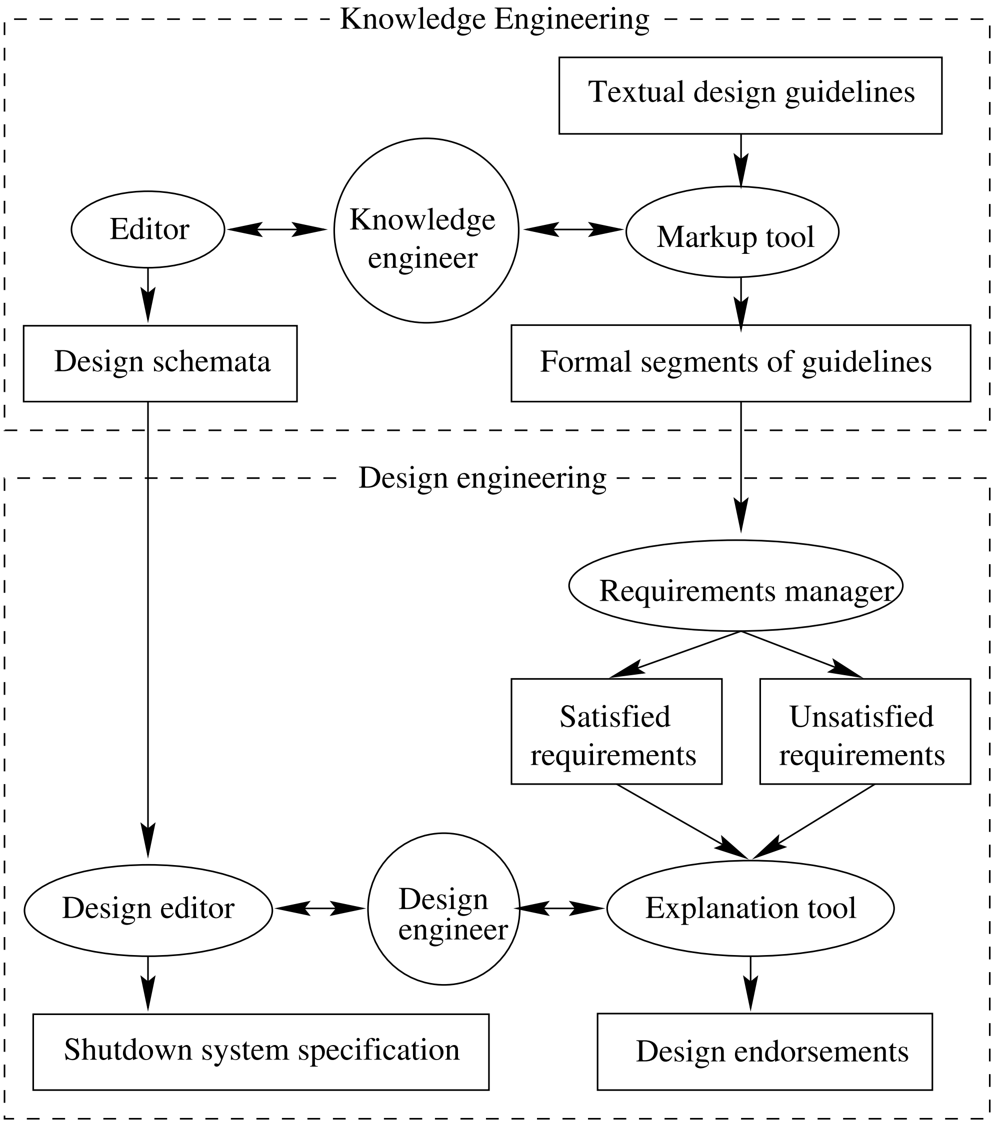


Investment in early automation pays dividends in maintenance.

Same Pattern, Different Domain



Linking Components to Rationale

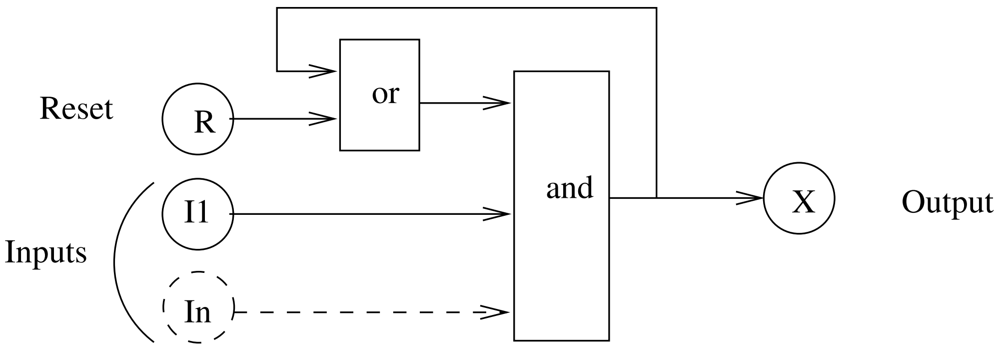


Relating Components to Requirements

Precondition (instantiates component):

$$output(X) \wedge$$
$$local\_reset(R, X) \wedge$$
$$\{I_1, \cdots, I_n\} = \{I | expected\_trip(I, X)\}$$

Component (added to the design):



Properties (connect to requirements):

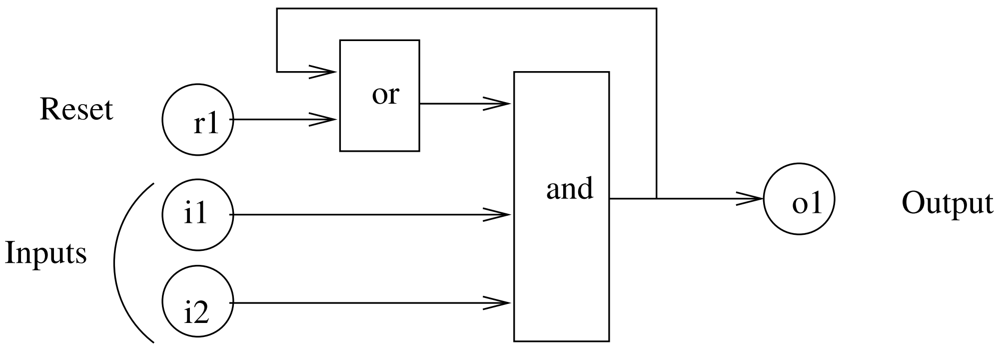
$$I \in \{I_1, \cdots, I_n\} \rightarrow causes(tripped(I), shutdown(X))$$
$$causes(shutdown(X) \wedge not\_reset(X), latched(X))$$

An Instance

Precondition (instantiated):

$$output(o1) \wedge$$
$$local\_reset(r1, o1) \wedge$$
$$\{i1, i2\} = \{I | expected\_trip(I, o1)\}$$

Component (added to the design):



Properties (connect to requirements):

$$I \in \{i1, i2\} \rightarrow causes(tripped(I), shutdown(o1))$$
$$causes(shutdown(o1) \wedge not\_reset(o1), latched(o1))$$

A Closed-World Proof

"All trip demands cause shutdown of the appropriate outputs".

Precondition:

$output(o1) \wedge expected\_trip(i1, o1) \wedge expected\_trip(i2, o1)$

Property (from instance of component):

$I \in \{i1, i2\} \rightarrow causes(tripped(I), shutdown(o1))$

Show:

$$\begin{gathered}\forall X, I.output(X) \wedge expected\_trip(I, X) \rightarrow \\ causes(tripped(I), shutdown(X))\end{gathered} \tag{1}$$

Expression 1 is equivalent to:

$$\neg \left(\begin{array}{l} \exists X, I.output(X) \wedge expected\_trip(I, X) \wedge \\ \neg causes(tripped(I), shutdown(X)) \end{array} \right) \tag{2}$$

Which is established by closed-world negation.

Proofs Connect Designs to Guidelines

This means (in theory) that:

- We can review each proof to check whether it is plausible.
- We can give points of failure for failed proofs.
- We can use the proofs as points of entry to the textual requirements.

System Modelling Assumptions

What's the right formal representation for our reset component ?

Is it this:

$$
\begin{array}{rcl}
activation(X,T,V) & \leftarrow & previous(T,T_p) \wedge \\
 & & activation(X,T_p,V_0) \wedge \\
 & & activation(R,T_p,V_r) \wedge \\
 & & or\_gate(\{V_0,V_r\},V_p) \wedge \\
 & & \left(\begin{array}{c} activation(I_1,T_p,V_1) \wedge \\ \vdots \\ activation(I_n,T_p,V_n) \end{array} \right) \wedge \\
 & & and\_gate(\{V_p,V_1,\cdots,V_n\},V), \\
activation(X,T_i,V_i) & \leftarrow & initial\_activation(X,T_i,V_i)
\end{array}
$$

or is it more sophisticated (*e.g* time lags) ?

Property Verification Assumptions

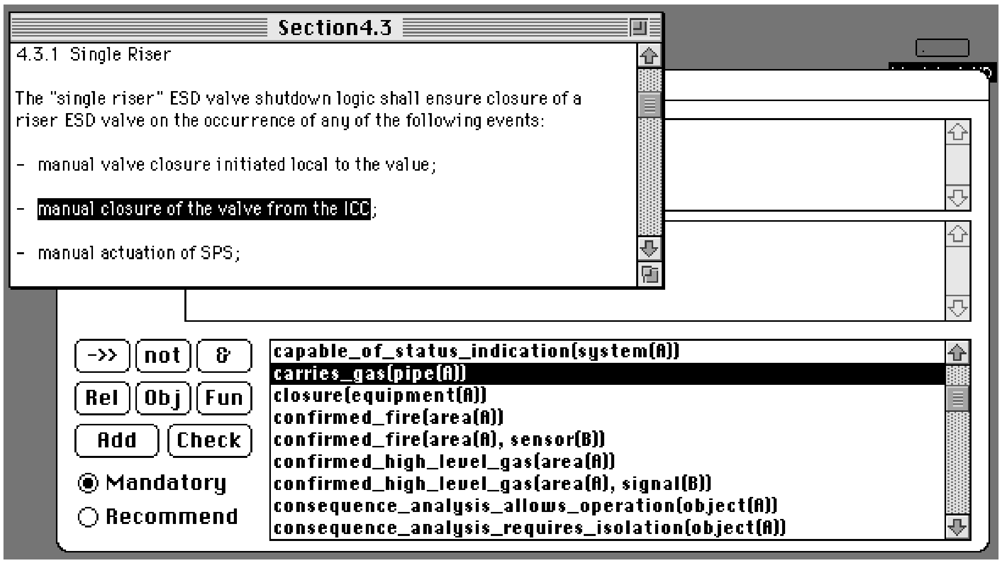
Properties are instantiated along with components.

We could either:

- trust that they hold for any instance of the component or
- prove that they hold every time they are used.

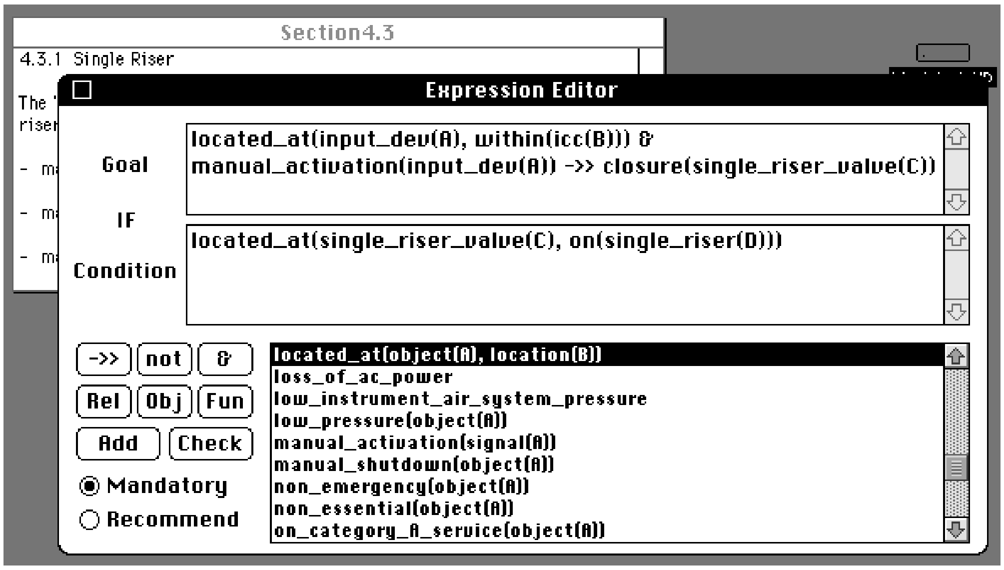
Ontological Assumptions (1/3)

All proofs depend on a formal connection to the textual requirements, so these must be marked up by an engineer . . .



Ontological Assumptions (2/3)

...and the formal names are combined using logical connectives ...



Ontological Assumptions (3/3)

...so there had better be a small number of formal names

Section	Reqs	Sorts	Functions	Relations
Section 4.1	60	130	13	47
Section 4.2	67	130	13	52
Section 4.3	109	137	14	59

and some concepts will not be represented formally.

Adequate Coverage Assumption

If we haven't represented everything formally then we are using proof as a "softer" form of support, using proof as a route into the textual requirements but not as an alternative to them. We need to check that those routes give access in the right way for engineers.

Proof Transparency Assumptions (1/2)

To understand why

$$\forall X, I.output(X) \wedge expected\_trip(I, X) \rightarrow causes(tripped(I), shutdown(X))$$

is established engineers need some education in formal proof, but only enough to check the proof retrospectively.

Proof Transparency Assumptions (2/2)

…but greater insight may be required to correct failed proofs, where the point of failure may not be the best place to begin correction.

Plausible Idealisation Assumptions

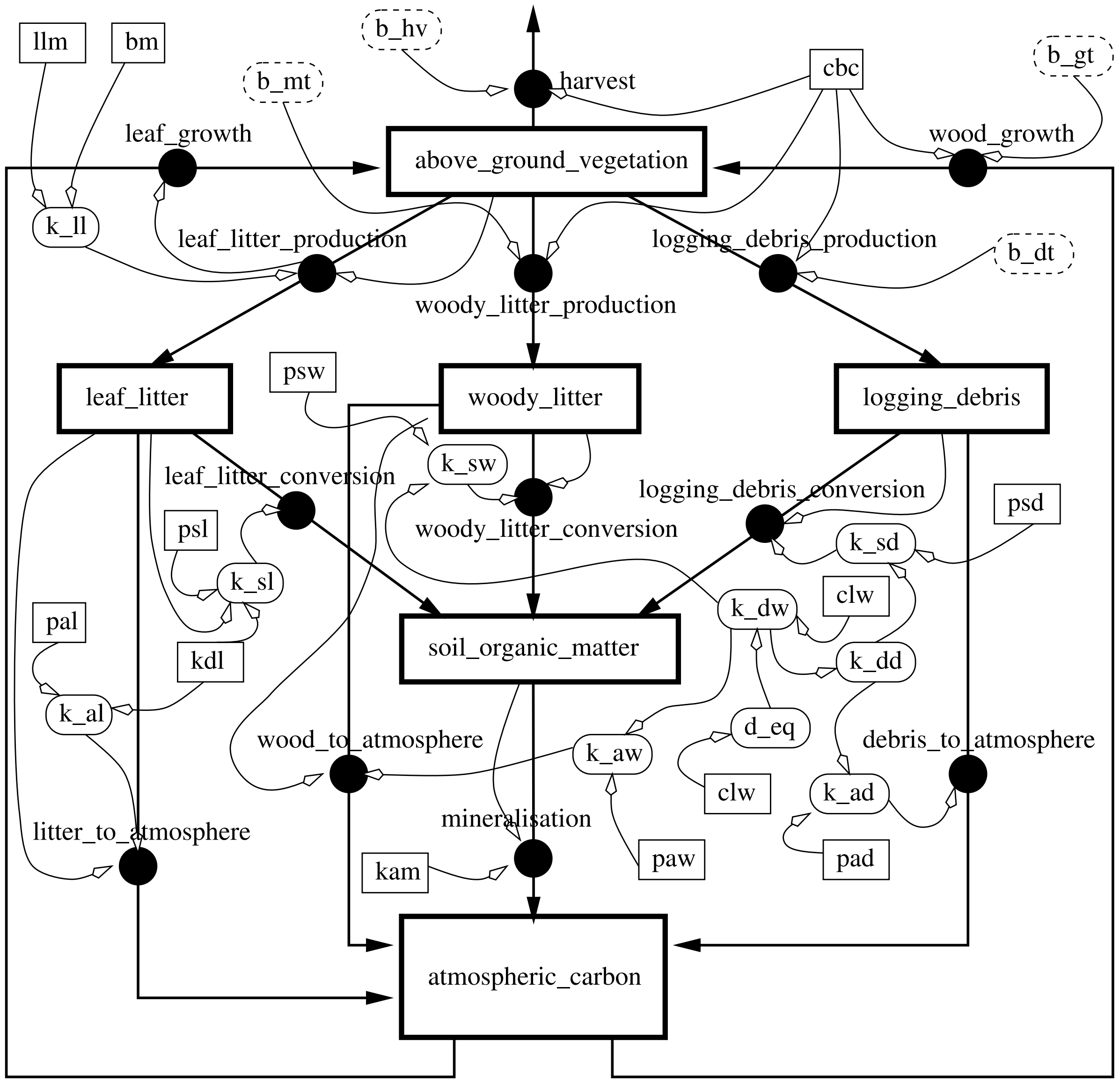
Our relay example is a model of a shutdown component. It gives an idealised description of a system which will be built using relays, *etc.* A specification for the system will have relays in it but our links to the requirements are through proofs at the level of the model. We must assume that either:

- the proofs at model level map straightforwardly to proofs at specification level or
- the properties we prove of the model are the right ones to inform specification engineers.

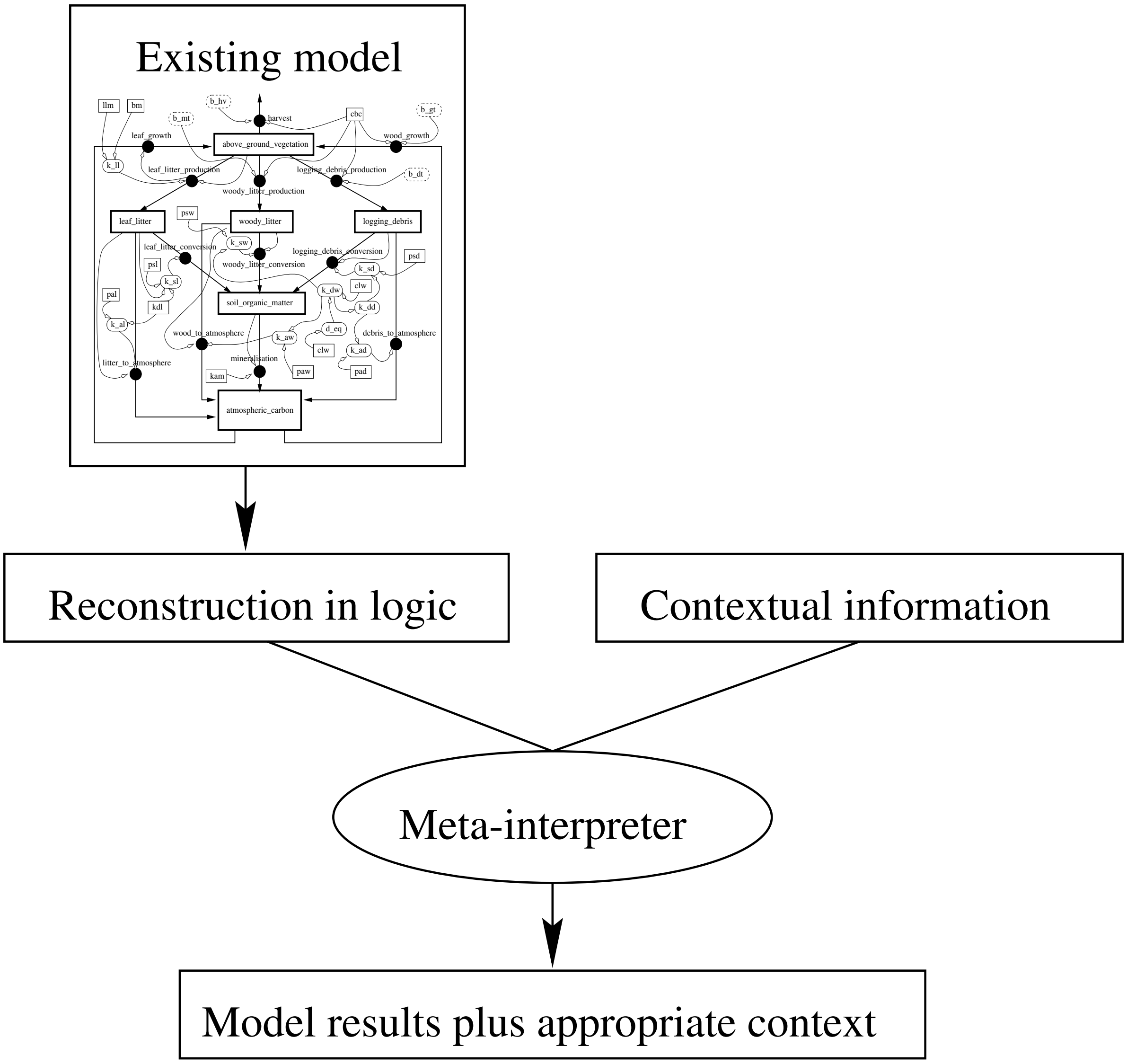
Safety Assumptions Summarised

1. Modelling (at model level) must suit form of proof (at requirements level).
2. Properties predefined for components must hold for all instances ...
3. or they must be re-verified for each use.
4. The number of formal names used must be small ...
5. so large textual requirements aren't fully formalised ...
6. and we can't throw away the text.
7. Engineers must know enough about proof to check proofs retrospectively ...
8. and may need to know enough to hypothesise causes of failure in proofs.
9. The idealised proofs relevant to system models must be consitently interpreted by those building specifications.

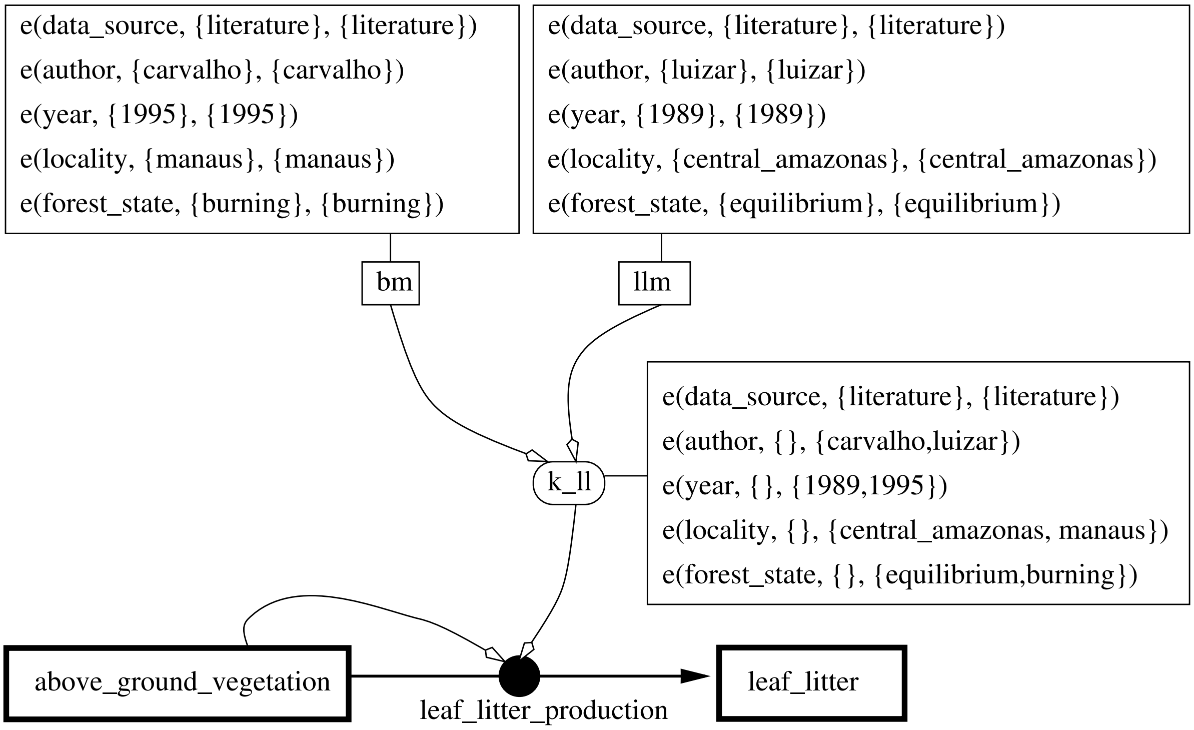
Tracing Context can be Complex



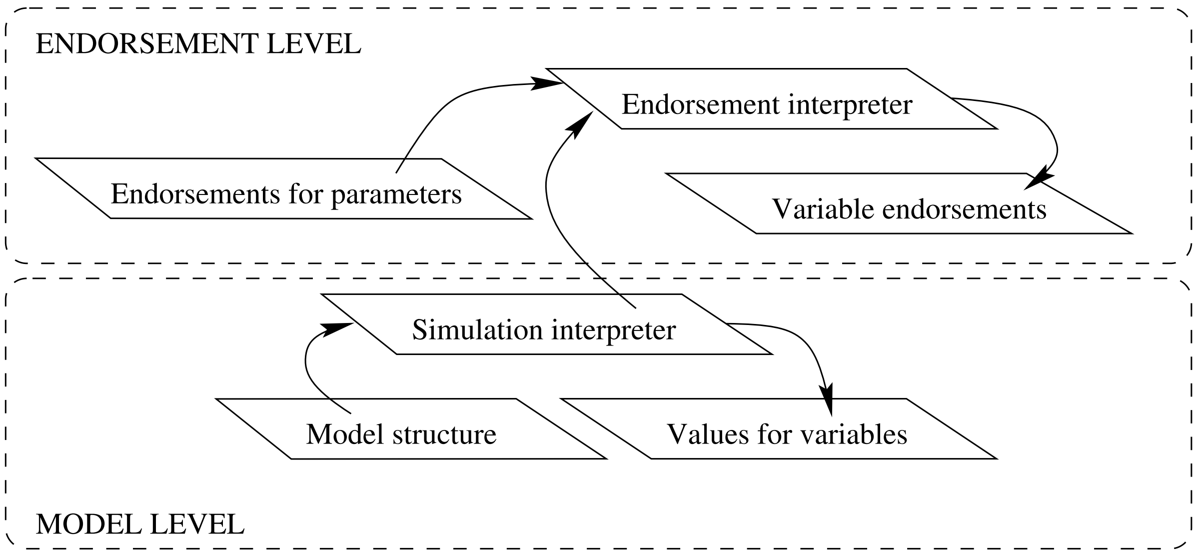
Automating Traceability



Context Related to Structure

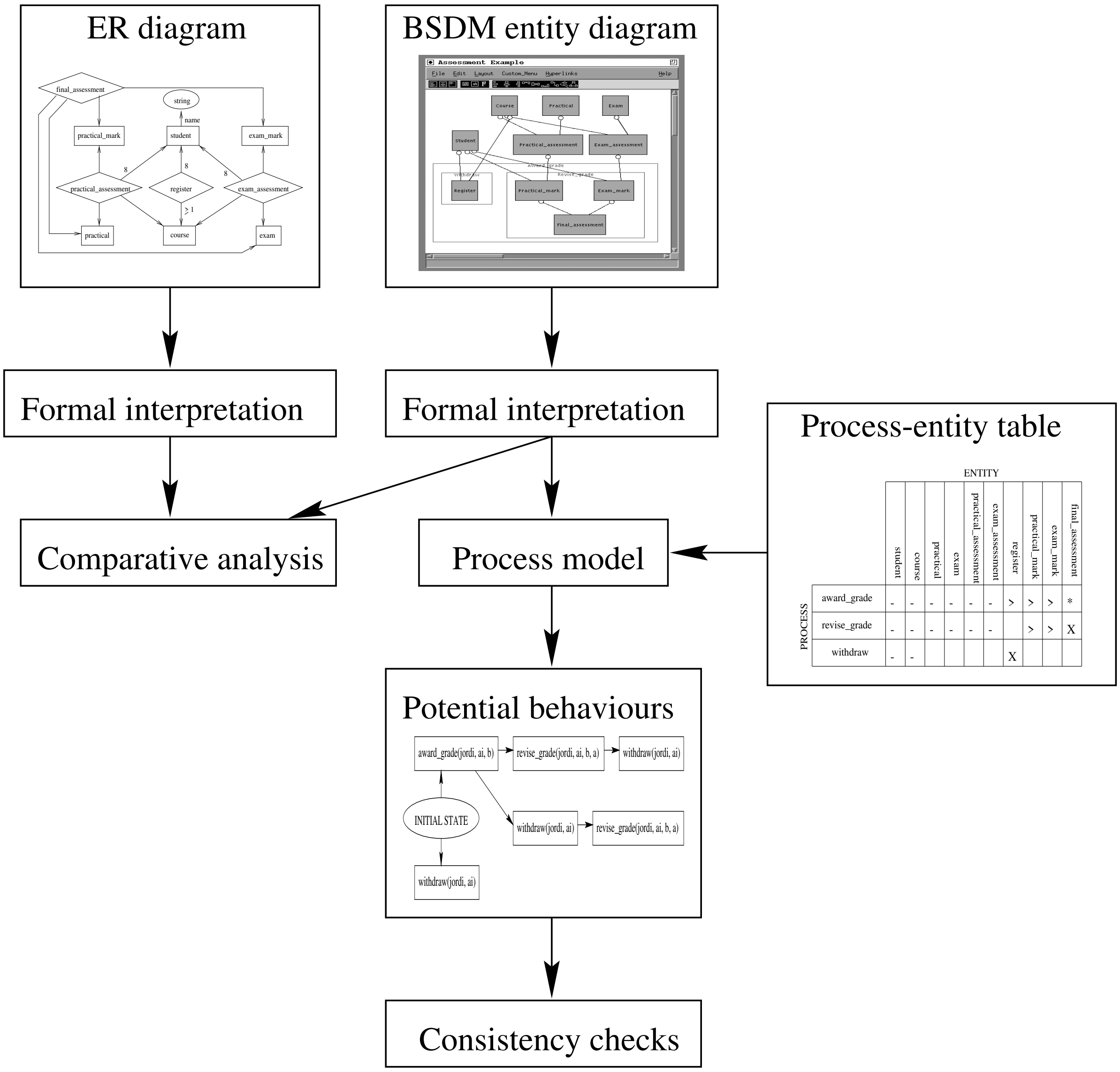


A Reflective Tracing Architecture



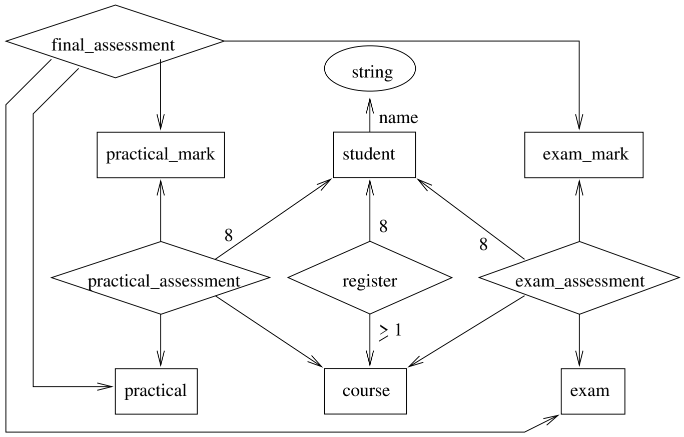
Connecting to Established Lifecycles

IBM's Business System Development Method

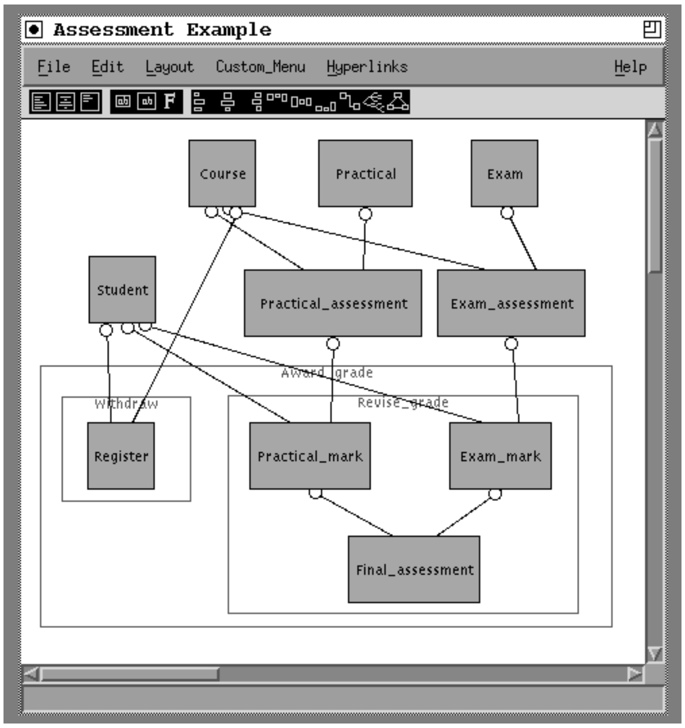


Comparing Notations (1)

How does an ER model like this



compare to a BSDM entity model like this?



Comparing Notations (2)

View the ER model formally as:

$$student \subseteq entity$$
$$course \subseteq entity$$
$$practical \subseteq entity$$
$$exam \subseteq entity$$
$$practical\_mark \subseteq entity$$
$$exam\_mark \subseteq entity$$

$$register(X_s, X_c) \rightarrow X_s \in student \wedge X_c \in course$$
$$\forall X_s . |\{X_c : register(X_s, X_c)\}| = 8$$
$$\forall X_c . |\{X_s : register(X_s, X_c)\}| \geq 1$$

$$\begin{aligned} practical\_assessment(X_s, X_c, X_p, X_m) \rightarrow \quad & X_s \in student \wedge \\ & X_c \in course \wedge \\ & X_p \in practical \wedge \\ & X_m \in practical\_mark \end{aligned}$$

$$\forall X_s . |\{(X_c, X_p, X_m) : practical\_assessment(X_s, X_c, X_p, X_m)\}| = 8$$

$$\begin{aligned} exam\_assessment(X_s, X_c, X_e, X_m) \rightarrow \quad & X_s \in student \wedge \\ & X_c \in course \wedge \\ & X_e \in exam \wedge \\ & X_m \in exam\_mark \end{aligned}$$

$$\forall X_s . |\{(X_c, X_e, X_m) : exam\_assessment(X_s, X_c, X_e, X_m)\}| = 8$$

$$\begin{aligned} final\_assessment(X_s, X_c, X_p, X_m) \rightarrow \quad & X_s \in student \wedge \\ & X_c \in course \wedge \\ & X_p \in practical \wedge \\ & X_m \in exam\_mark \end{aligned}$$

$$name(X_e, X_v) \in attribute \rightarrow X_e \in student \wedge X_v \in string$$

Comparing Notations (3)

View the SDM entity model formally as:

$$\begin{gathered} student \subseteq entity \\ course \subseteq entity \\ practical \subseteq entity \\ exam \subseteq entity \end{gathered}$$

$$\begin{array}{l}
register(X_s, X_c) \in register \rightarrow \\
\quad X_s \in student \wedge \\
\quad X_c \in course \\
practical_assessment(X_c, X_p) \in practical_assessment \rightarrow \\
\quad X_c \in course \wedge \\
\quad X_p \in practical \\
exam_assessment(X_c, X_e) \in exam_assessment \rightarrow \\
\quad X_c \in course \wedge \\
\quad X_e \in exam \\
practical_mark(X_s, X_{pa}) \in practical_mark \rightarrow \\
\quad X_s \in student \wedge \\
\quad X_{pa} \in practical_assessment \\
exam_mark(X_s, X_{ea}) \in exam_mark \rightarrow \\
\quad X_e \in student \wedge \\
\quad X_{ea} \in exam_assessment \\
final_assessment(X_{em}, X_{pm}) \in final_assessment \rightarrow \\
\quad X_{em} \in exam_mark \wedge \\
\quad X_{pm} \in practical_mark
\end{array}$$

Models as Constraints on Design (1)

PROCESS \ ENTITY	student	course	practical	exam	practical_assessment	exam_assessment	register	practical_mark	exam_mark	final_assessment
award_grade	-	-	-	-	-	-	>	>	>	*
revise_grade	-	-	-	-	-	-		>	>	X
withdraw	-	-					X			

$$originates(award\_grade(S, C, G), final\_assessment)$$
$$requires(award\_grade(S, C, G), exam\_mark)$$
$$requires(award\_grade(S, C, G), practical\_mark)$$
$$requires(award\_grade(S, C, G), register)$$
$$requires(revise\_grade(S, C, G, R), exam\_mark)$$
$$requires(revise\_grade(S, C, G, R), practical\_mark)$$
$$changes(revise\_grade(S, C, G, R), final\_assessment)$$
$$changes(withdraw(S, C), register)$$

Models as Constraints on Design (2)

Given:

- Process name P.
- Set of originated items Or
- Set of required items Re.
- Set of changed items Ch.
- Set of deleted items De.

$$
\begin{aligned}
& scope\_discrepancy(P, Or, Re, Ch, De) \leftarrow \\
& \qquad originates(P, T) \wedge \\
& \qquad not(E \in Or \wedge valid\_state\_type(E, T)) \\
& scope\_discrepancy(P, Or, Re, Ch, De) \leftarrow \\
& \qquad requires(P, T) \wedge \\
& \qquad not(E \in Re \wedge valid\_state\_type(E, T)) \\
& scope\_discrepancy(P, Or, Re, Ch, De) \leftarrow \\
& \qquad changes(P, T) \wedge \\
& \qquad not \left(\begin{array}{c} (E \in De \ \vee \ (E, E') \in Ch)) \wedge \\ valid\_state\_type(E, T) \end{array} \right)
\end{aligned}
$$

Formal Pragmatics (1)

A stucture editor.

```
path(X) if
    <1> PERFORM A TEST ON [X]

path([H|T]) if
    <2> PERFORM A TEST ON [H]
    path(T)
```

```
path(X,V) if
    <1> PERFORM A TEST ON [X]

path([H|T],V) if
    <2> PERFORM A TEST ON [H]
    path(T,V)
```

```
path(X,V,Final) if
    <1> PERFORM A TEST ON [X]
    <2> INSTANTIATE Final

path([H|T],V,Final) if
    <3> PERFORM A TEST ON [H]
    path(T,V,P)
    <4> OBTAIN Final FROM [P]
```

```
path(X,V,Final) if
    X = [V|T]
    Final = T

path([H|T],V,Final) if
    path(T,V,P)
    Final = [H|P]
```

Formal Pragmatics (2)

Intuitively correct combination.

$$\begin{array}{ll} path([X|T], X, []) & dist([X|T], X, 0) \\ path([H|T], X, [H|R]) \leftarrow & dist([H|T], X, N) \leftarrow \\ \quad path(T, X, R) & \quad dist(T, X, M) \wedge \\ & \quad N \; is \; M + 1 \end{array}$$

$$\Downarrow \qquad\qquad\qquad \Downarrow$$

$$path\_dist(L, X, P, D) \leftarrow path(L, X, P) \wedge dist(L, X, D)$$

$$\Downarrow$$

$$\begin{array}{l} path\_dist([X|T], X, [], 0) \\ path\_dist([H|T], X, [H|R], N) \leftarrow \\ \quad path\_dist(T, X, R, M) \wedge \\ \quad N \; is \; M + 1 \end{array}$$

Formal Pragmatics (3)

Logically correct combination.

$$
\begin{array}{l}
path\_dist([X|T], X, [], 0) \\
path\_dist([H|T], X, [H|R], N) \leftarrow \\
\quad path\_dist(T, X, R, M) \wedge \\
\quad N \; is \; M + 1 \\
path\_dist([H|T], X, [], N) \leftarrow \\
\quad dist(T, X, M) \wedge \\
\quad N \; is \; M + 1 \\
path\_dist([X|T], X, [X|R], 0) \leftarrow \\
\quad path(T, X, R)
\end{array}
$$

For goal: $path\_dist([a, b, c, d, c], c, P, D)$ gives:

$$
\begin{array}{rl}
P = [a, b] & D = 2 \\
P = [a, b, c, d] & D = 4 \\
P = [a, b] & D = 4 \\
P = [a, b, c, d] & D = 2
\end{array}
$$

Last two of these normally aren't wanted but are consistent with the join specification.