Prozeßbeschreibung und Modellierungsrichtlinien für den Analyseprozeß

In diesem Dokument soll zunächst das Vorgehen für den Analyseprozeß im allgemeinen und dann die Verwendung der zur Verfügung gestellten Modellierungselemente erklärt werden.

Die Modellierung des Prozesses entstand in Zusammenarbeit mit dem Teilprojekt B1 des SFB 501 an der Universität Kaiserslautern.

Wichtig: Kapitel 4, die Richtlinien für die SDL-Modellierung werden noch überarbeitet!

1 Grundprinzipien der Analyse

Startpunkt des Analyse ist eine Problembeschreibung, durch die das Verhalten des gewünschten Systems in Form von Needs abstrakt dargestellt wird. Ziel ist es, ein ausführbares Modell zu erstellen, das das (funktionale) Verhalten des zu entwerfenden Systems darstellt.

Das gesamte Vorgehen der Analyse basiert auf einer hierarchischen Unterteilung des zu entwerfenden Systems. Die dabei entstehenden einzelnen Teile des Systems (im folgenden Objekte) durchlaufen während der Analyse verschiedene Phasen. Dieses Durchlaufen kann größtenteils zeitlich getrennt erfolgen, d.h. es existieren einige wenige Punkte, zu denen alle Objekte einen bestimmten Zustand erreicht haben müssen. Diese Punkte dienen der Synchronisation, die bei einer verteilten Entwicklung immer wieder gewährleistet sein muß.

Die erste Aufgabe in einer Analyse ist die Beschäftigung mit dem Problem. Dazu muß die Problembeschreibung verstanden werden und es wird ein bestehendes ProjectDictionary betrachtet und gegebenenfalls erweitert.

Als nächstes steht die Ermittlung einer geeigneten Hierarchie im Mittelpunkt. Ausgangspunkt dafür kann ein Modell des Gebäudes sein, auf das sich die Problembeschreibung bezieht. Es entsteht eine Typ-Instanzen-Hierarchie, die die auftretenden Objekttypen festlegt.

An diese Phase gliedert sich die Entwicklung der einzelnen Objekttypen an. Dabei werden Aufgaben (Tasks) den Objekttypen zugeordnet und sie werden schließlich in SDL-Modelle überführt.

Die in der Entwicklung zu treffenden Entscheidungen sind einzelnen Prozeßschritten zugeordnet, die im einzelnen in Kapitel 1.1 erklärt werden.

Die Entscheidungen werden durch Operationen in die einzelnen Dokumente eingebracht. Diese Operationen werden in Kapitel 1.2 im einzelnen kurz dargestellt.

Um einen Überblick über den aktuellen Entwurfsstand zu gewährleisten, wird die Entwicklung der Objekte in verschiedene Phasen eingeteilt. Diese Phasen werden in Kapitel 1.3 beschrieben.

1.1 Die Prozeßschritte

Die gesamte Analyse wird in verschiedene Schritte eingeteilt. Diese Schritte unterscheiden sich zu einem darin, welche Entwicklungsentscheidungen in ihnen getroffen werden, zum anderen natürlich darin, welche Eingabedokumente sie benötigen und welche Dokumente sie erzeugen bzw. modifizieren. Diese Produktabhängigkeiten sind in Abbildung 1 (für den ersten

Teil) und Abbildung 2 (für den zweiten Teil) dargestellt. Der erste Teil basiert im wesentlichen

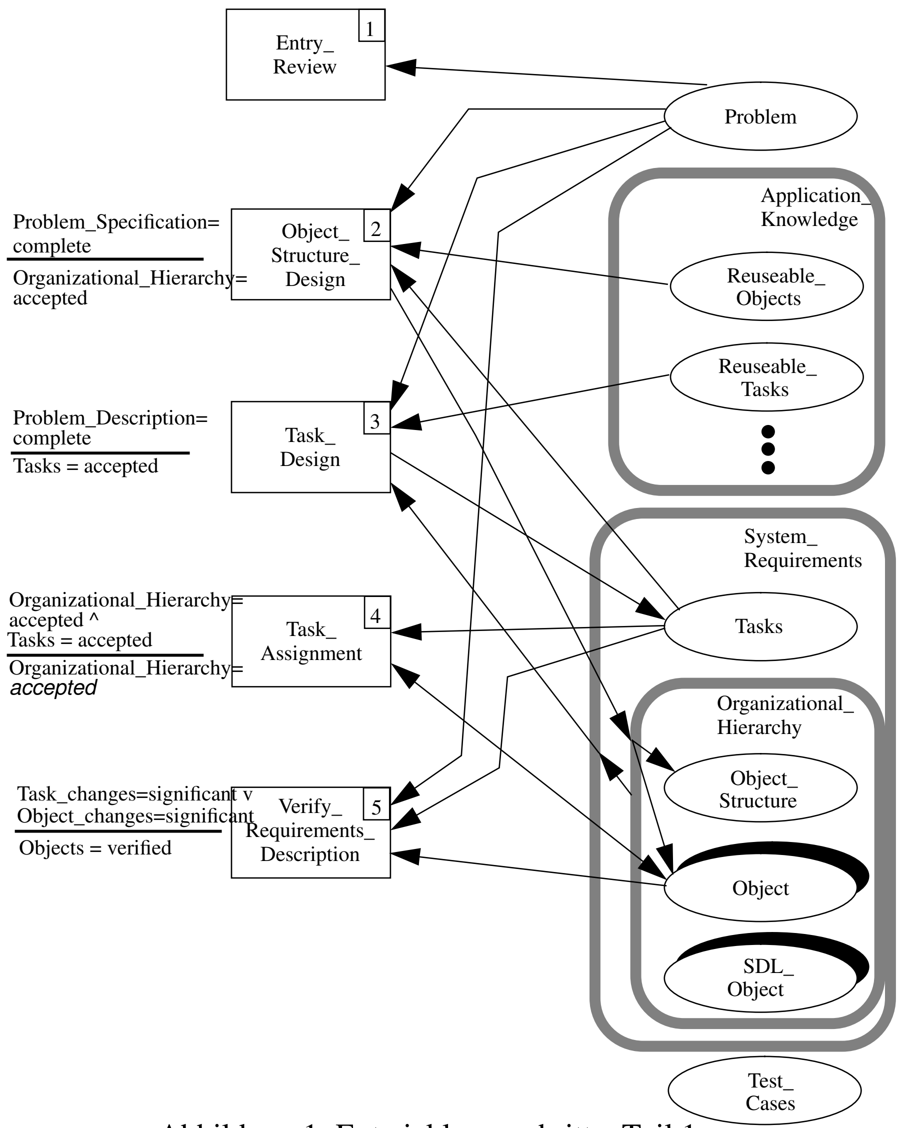

Abbildung 1: Entwicklungsschritte, Teil 1

auf Beschreibungen in natürlicher Sprache und ist damit weniger formal als der zweite Teil, in dem eine formale Sprache (SDL) benutzt wird. Damit ist die Ausgabe der "späten" Prozeßschritte maschinell interpretierbar.

EntryReview

Start des Analyseprozesses ist ein Verstehen des Problems. Dazu gehört in erster Linie ein Beschäftigen mit der Problembeschreibung. Weiterhin wird in diesem Schritt schon das Dictionary betrachtet, das die in der Problembeschreibung auftretende oder in der Domäne wichtige Begriffe erläutert.

ObjectStructureDesign

Eingabe in diesen Schritt bildet die Problembeschreibung. Es wird eine erste Version der Objektstruktur erzeugt, die gewährleisten sollte, daß in den nachfolgenden Schritten eine sinnvolle Zuordnung von Tasks zu Objekttypen und von Objekttypen zu Entwerfern durchgeführt werden kann. Ausgabe ist entweder ein Dokument, das die Objektstruktur beschreibt (Produkt *ObjectStructure*), oder eine Reihe von Dokumenten, die jeweils einen einzelnen Objekttyp beschreiben. Ist die Ausgabe das umfassende Dokument *ObjectStructure*, muß diese Struktur im weiteren Verlauf in die Dokumente der einzelnen Objekttypen überführt werden.

TaskDesign

Hier werden die wesentlichen Tasks definiert. Sie müssen noch keinen Objekttypen zugeordnet sein, vielmehr bedeutet dies eine erste Verfeinerung der (ebenfalls nicht Objekten zugeordneten) Needs. Eingabe ist wiederum die ProblemDescription, Ausgabe ist ein Dokument, das alle dort gefundenen Tasks enthält.

Wird dieser Prozeßschritt bei der Bearbeitung eines einzelnen Objekttyps durchgeführt, werden in diesem Schritt zu den Tasks die Strategien angegeben, die eine erste Realisierungsmöglichkeit angeben. Weiterhin besteht dann keine Notwendigkeit zur Durchführung des Schritts *TaskAssignment*, da der definierte Task ja schon einem Objekttyp zugeordnet ist.

TaskAssignment

Nach Erzeugung der Objektstruktur und der Tasks können die Tasks aus dem globalen Dokument *Tasks* Objekttypen zugewiesen werden. Diese Zuweisung passiert in diesem Schritt, und erweitert die Objekttypen, die in einzelnen Dokumenten beschrieben sind.

VerifyRequirementDescription

Nach der Ersterstellung oder wenn größere Änderungen bzw. Ergänzungen in den ersten drei Prozeßschritten stattfanden, ist es sinnvoll, eine anschließende Verifikation durchzuführen. Dazu wird die RequirementDescription (die Objekte mit ihren Tasks) auf Konsistenz und Korrektheit gegenüber der Problembeschreibung überprüft. Es entsteht eine Fehlerliste, die innerhalb dieses Schritts abgearbeitet wird (siehe Kapitel 3.5).

Die bis hierhin dargestellten Prozeßschritte beziehen sich auf den ersten Teil der Entwicklung.

Die Prozeßschritte des zweiten Teils, also der Überführung der informellen Dokumente in SDL-Modelle, wird in der folgenden Abbildung beschrieben.

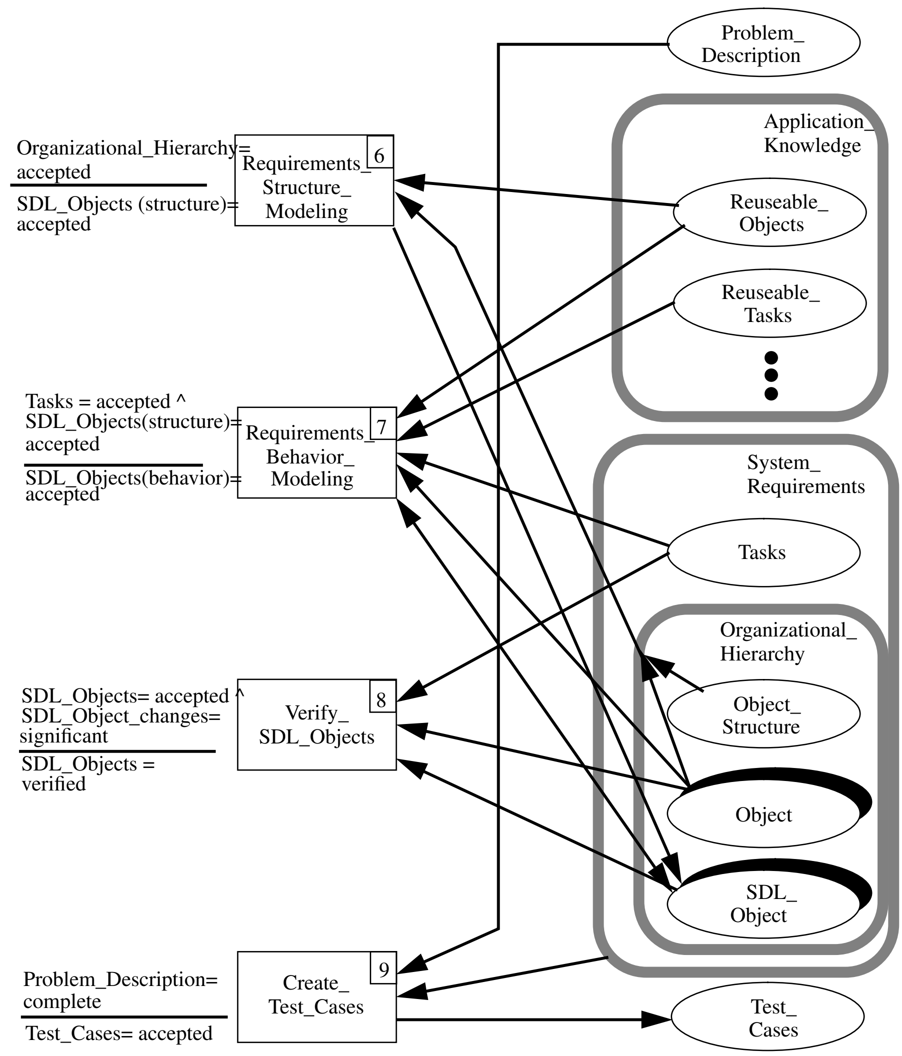

Abbildung 2: Entwicklungsschritte, Teil 2

RequirementsStructureModeling

Der erste Schritt bei der Überführung der informellen Beschreibung in SDL-Modelle besteht in der Erstellung der Struktur der SDL-Modelle. Es entstehen die *SDL-Objects*, die durch den nachfolgenden Schritt um die Zustandsgraphen erweitert werden.

RequirementsBehaviorModeling

Dieser Schritt beinhaltet die Modellierung der Zustandsgraphen für die Objekttypen, deren Struktur bereits in SDL beschrieben wurde.

VerifySDL-Objects
Hier wird der SDL-Entwurf durch Inspektion oder lokales Prototyping überprüft. Näheres dazu findet sich in Kapitel 3.4.

CreateTestCases
Eingabe sind neben der Problembeschreibung prinzipiell alle in der Entwicklung auftretenden Dokumente. Es muß sichergestellt werden, daß alle in der Problembeschreibung genannten Needs für typische und extreme Fälle gestestet werden (black-box-testing). Darüberhinaus kann in Form eines white-box-testings die Realisierung, d.h. die Strategien, überprüft werden.

1.2 Operationen auf den Dokumenten

Durch die Operationen werden die Entwicklungsentscheidungen, die in den Schritten aus Kapitel 1.1 getroffen wurden, in die Dokumente eingebracht. Die verschiedenen Operationen, die auf den Dokumenten erlaubt sind, lassen sich grob in vier Gruppen einteilen: erzeugende, verändernde, verbessernde und überprüfende. Zusätzlich existiert noch die Operation zum Konfigurieren eines Objekts. Die Bezeichnungen in Klammern geben das Kürzel an, das innerhalb der Entwicklung benutzt wird (siehe Kapitel 2.1).

Desweiteren wird ein Reuse-Ansatz (siehe Abbildung 3) vorgestellt, der immer vor der Ausführung der Operationen *create* und *extend* durchlaufen werden sollte.

1.2.1 Erzeugende Operationen

- **create (cre)**
  Die “einfachste” Art, ein neues Dokument anzulegen. Eventuell werden hier andere Dokumente als Eingabe benötigt. Dies kann z.B. ein Dokument, das die Struktur des Systems beschreibt, sein.
- **create Alternative (cra)**
  Diese Operation geht von einem bestimmten Dokument aus und erzeugt als Alternative ein neues Dokument. Beschreibt das Ausgangsdokument einen Objekttyp, so muß der alternative Objekttyp einen neuen, eindeutigen Namen besitzen.

1.2.2 Verändernde Operationen

- **refine (ref)**
  Durch diese Operation werden Verbesserungen an einem Dokument vorgenommen. Eventuelle, daraus resultierende Änderungen an anderen Dokumenten resultieren in einer Change-Liste bei dem jeweiligen Objekt.
- **extend (ext)**
  Durch diese Operation werden Erweiterungen an einem Dokument vorgenommen.

1.2.3 Verbessernde Operationen

- **change (cha)**
  Prinzipiell können in jedem Entwicklungszustand und in jeder Operation bei einem Dokument Changes erzeugt werden, die, einer nach dem anderen, abgearbeitet werden müssen. Dazu dient diese Operation, die die Changeliste als Eingabe erhält und deren Einträge bearbeitet.
- **correct (cor)**
  Wurden in anderen Operationen Fehler in einem Objekt gefunden, werden diese in die Errorliste eingetragen und durch die correct-Operation behoben.

1.2.4 Überprüfende Operationen

- **review (rev)**
  Diese Operation beinhaltet eine Inspektion des Objekts und erzeugt eine Fehlerliste.
- **test (test)**
  Das Testen ist eine Art der Validierung und überprüft die Korrektheit durch eine maschinelle Interpretation der Dokumente. Die sind im Allgemeinen SDT-Dokumente. Diese können auf Syntax und Semantik überprüft werden oder es kann ein Prototyp aus ihnen generiert werden. Hierbei werden ebenfalls Fehler gefunden und in eine Liste eingetragen.

1.2.5 Konfigurieren

- **productConfigure (prdCfg)**
  Durch die hierarchische Gliederung eines Systems entsteht eine Aggregationshierarchie,

bei der unter anderem von einem Typ aus auf andere Typen verwiesen wird. Zu bestimmten Zeitpunkten (spätestens, wenn ein Prototyp erzeugt wird), muß festgelegt werden, welche Version eines Typs an welcher Stelle eingesetzt werden soll. Diese Festlegung bezeichnet man als Produktkonfiguration. Sie wird i.A. bottom-up im Hierarchiebaum durchgeführt, sodaß immer mit einer bestehenden Produktkonfiguration das nächste höhere Objekt konfiguriert wird.

- **processConfigure (prcCfg)**
  Der Projektmanager kann jederzeit Konfigurationen von konsistenten Dokumenten bilden. Zumindest zu Abschluß jeder Phase sollte solch eine Prozeßkonfiguration gebildet werden. Diese in dieser Konfiguration enthaltenen Dokumente werden dann auf Konsistenz geprüft und eventuelle Fehler behoben. Diese drei Operationen (*prcCfg*, *rev*, *corr*) werden iteriert, bis kein Fehler mehr entdeckt wird.

1.2.6 Reuse

Hier wird kurz ein Reuse-Ansatz vorgestellt, der dazu dient, bei bestimmten Operationen bestehende Artefakte wiederzuverwenden.

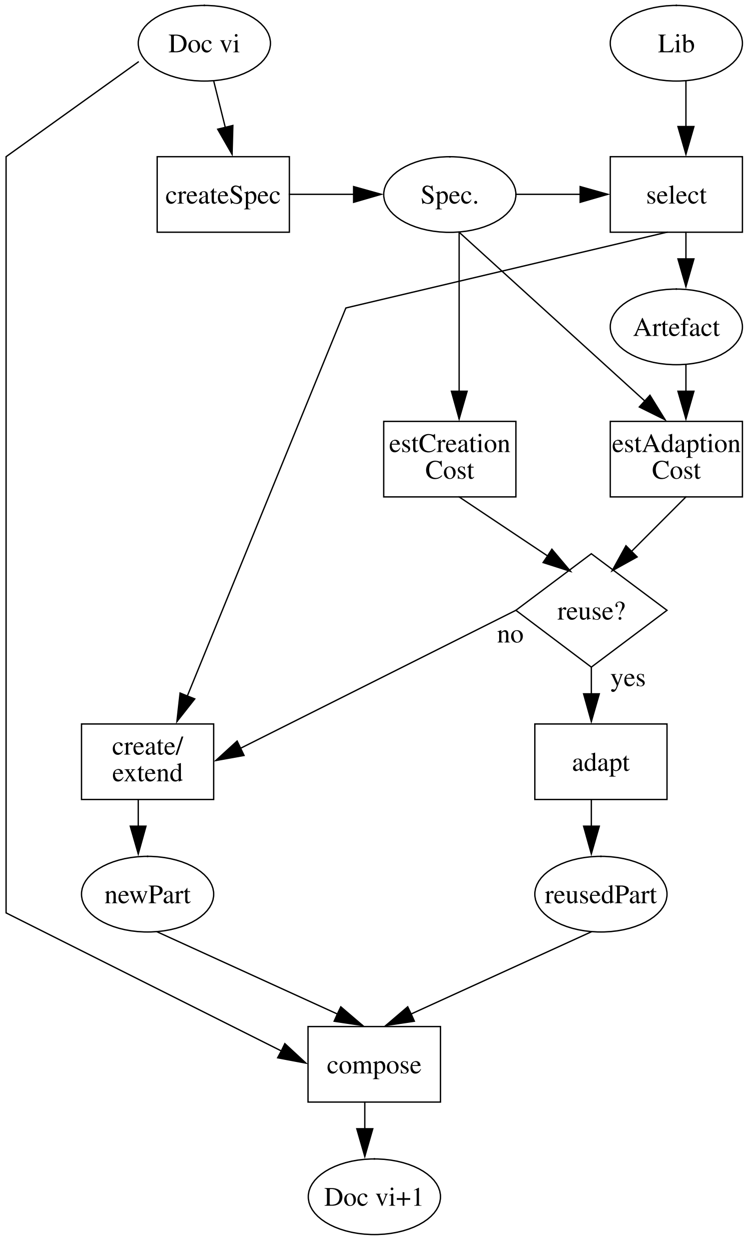

Abbildung 3: Reuse-Vorgehen

Die Operationen, bei denen ein mögliches Reuse geprüft wird, sind die Operationen *create* und *extend*. Die erste Aufgabe ist es, eine (kurze) Spezifikation zu erstellen (*createSpec*), die Eingabe in einen Selektionsprozeß (*select*) ist. Es wird eventuell ein Artefakt (*Artefakt*) zur potentiellen Wiederverwendung gefunden. Ist das nicht der Fall, Wird die Operation konventionell durchgeführt (*create/extend*), es entsteht ein neuer Teil eines Dokuments (oder ein neues Dokument bei einer create-Operation), das dann in die bestehenden Version eingefügt wird (*compose*).

Wurde bei dem Selektionsvorgang ein Artefakt gefunden, müssen zunächst die Kosten für eine Adaption (*estAdaptionCost*) bzw. für eine Neuerstellung (*estCreationCost*) abgeschätzt und miteinander verglichen werden. Dieser Vergleich ist Grundlage für die Entscheidung, ob wiederverwandt wird oder nicht (*reuse?*). Dementsprechend wird dann die Adaption (*adapt*) durchgeführt und man erhält ein Artefakt (*reusedPart*), daß nun wiederum in das bestehende Dokument eingebracht wird.

Ein spezieller Fall kann auftreten, wenn das gefundene Artefakt nicht vollständig die Spezifikation erfüllt. Dann ist sozusagen die reuse-Entscheidung (*reuse?*) ein "jein" und es wird sowohl ein *reusedPart* als auch ein *newPart* erstellt.

1.3 Die Phasen der Analyse

Die Definition verschiedener Phasen in einer Entwicklung erlaubt es, Dokumente, die konsistent zueinander sind, zu finden und zusammenzufassen. Konsistent hierbei bedeutet, daß die Dokumente mindestens die jeweils betrachtete Phase erreicht haben müssen. Sie können jedoch auch schon weiter in der Entwicklung fortgeschritten sein. Da unser Vorgehen auf einer schrittweisen Verfeinerung besteht, besteht die Entwicklung fast ausschließlich aus Erzeugungen, Erweiterungen bzw. Verbesserungen der Dokumente. Dementsprechend bleiben die Ergebnisse und Entwicklungsentscheidungen aus früheren Phasen auch in den späteren Phasen in den Dokumenten erhalten.

Alle Phasen werden von den verschiedenen Dokumenten zeitlich voneinander getrennt durchlaufen. Jedoch ist die Reihenfolge der Phasen, die ein Dokument durchläuft, festgelegt. Hat es eine Phase beendet, steht es zur Bildung einer entsprechenden Konfiguration zur Verfügung. Diese Zusammenfassung von Dokumenten, die denselben Entwicklungszustand erreicht haben, dient der Synchronisation der Entwicklung und kann bei einem eventuellen Rückschritt als neuer Aufsatzpunkt genutzt werden. Nach Bildung einer Konfiguration werden alle Dokumente dieser Konfiguration verifiziert und eventuelle Fehler behoben.

Im folgenden werden die einzelnen Phasen in ihrer Reihenfolge beschrieben. Die Kürzel in Klammern werden wieder zur Zuordnung innerhalb der Analyse benutzt (siehe Kapitel 2.1).

1.3.1 StructureCreationPhase (StrCrePh)

In dieser Phase wird eine erste, hierarchische Aufteilung des Systems ermittelt. Ausgangspunkt dabei ist i. A. eine Modell des Gebäudes, das in der BldDsc beschrieben ist. Es entsteht eine Anzahl von Objekttypen, die ausschließlich über die Aggregationsbeziehung in Verbindung stehen. Zur Bildung dieser Hierarchie können die in der PrbDsc gegebenen Needs schon betrachtet, in Tasks aufgespalten und Objekten zugewiesen werden.

Diese Phase wird verlassen, wenn eine, zu dem Zeitpunkt vollständige Hierarchie entstanden ist.

1.3.2 RequirementsDescriptionPhase(ReqDscPh)

Diese Phase dient dazu, das Verhalten der einzelnen Objekttypen in informeller Weise zu beschreiben. Weiterhin werden hier schon Signale definiert, die die Schnittstelle der Objekttypen darstellen.

Eingabe für diesen Schritt sind die Needs aus der Problembeschreibung, die hier einzeln betrachtet werden. Diese Phase kann verlassen werden, wenn mindestens ein Task mit seinen Strategien und benötigten bzw. erzeugten Signalen beschrieben wurde.

1.3.3 RequirementsModellingPhase (ReqModPh)

Die Überführung der textuellen Darstellung der Objekttypen in das formale SDL-Modell wird in dieser Phase abgehandelt. Zunächst wird die Struktur eines Objekttyps in SDL beschreiben, und dann die Zustandsgraphen aus den gegebenen Tasks mit ihren Strategien hergeleitet.

Ein Objekttyp verläßt diese Phase, wenn seine Struktur und sein verhalten vollständig in SDL modelliert wurde.

1.3.4 TestPhase (TestPh)

In dieser Phase wird das in den bisherigen Phasen entwickelte SDL-Modell gegen die TestCases getestet. Auftretende Fehler werden ebenfalls in dieser Phase behoben.

1.4 Zuordnung Operation zu Prozeßschritte

Die folgende Tabelle beschreibt, in welchen Prozeßschritten welche Operationen erlaubt sind.

Nr.	**Prozessname**	**Erlaubte Operationen innerhalb des Prozesses**	**Reuse-Prozess kann angewen-det werden**
1	Entry_Review	- keine -	nein
2	Object_ Structure_ Design	create, create Alternative, refine, extend, change, correct, productConfigure, processConfigure	ja
3	Task_ Design	create, create Alternative, refine, extend, change, correct, productConfigure, processConfigure	ja
4	Task_ Assignment	create, create Alternative, refine, extend, change, correct, productConfigure, processConfigure	ja
5	Verify_ Requirements_ Description	review, change, correct, productConfigure,processConfigure	nein
6	Requirements_ Structure_ Modeling	create, create Alternative, refine, extend, change, correct, productConfigure, processConfigure	ja
7	Requirements_ Behavior_ Modeling	create, create Alternative, refine, extend, change, correct test (für Komponententests), productConfigure, processConfigure	ja
8	Verify_ SDL_Objects	review, change, correct, productConfigure, processConfigure	nein
9	Create_ Test_Cases	create, create Alternative, refine, extend, change, correct, productConfigure, processConfigure	ja
10	Create_ Prototype	create, create Alternative, refine, extend, change, correct, productConfigure, processConfigure	ja
11	Prototype_ Test	test, change, correct, productConfigure, processConfigure	nein

2 Templates für die frühen Analysephasen

In diesem Kapitel sollen die Templates für die in Kapitel 1.3 beschriebenen Phasen vorgestellt werden. Ausnahme bildet der SDL-Entwurf, der in Kapitel 4 gesondert betrachtet wird. Die Entwicklungsentscheidungen werden in HTML dokumentiert. Am Ende jeden Abschnitts wird dann jeweils ein (unvollständiges) Beispiel gezeigt.

2.1 DocumentHistory-Template

Alle Versionen von Dokumenten werden durch die DocumentHistory zusammengefaßt. In die-

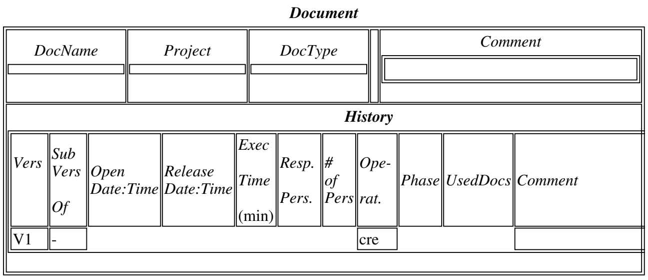

Abbildung 4: DocumentHistory-Template

ser Historie werden zum einen versionsübergreifenden Daten abgelegt.

- DocName: Der Name des Dokuments
- Project: Das Projekt, in dessen Kontext das Dokument entstanden ist
- DocType: Der jeweilige Dokumenttyp (PrbDsc, ObjStr, ..., siehe nachfolgende Kapitel)
- Comment: Ein kurzer Kommentar zur Entstehung des Dokuments

Die versionsspezifischen Verwaltungsdaten werden in die untere Tabelle eingetragen.

- Vers: Die Versionsnummer des jeweiligen Dokuments
- SubVersOf: Die Versionsnummer der Vorgängerversion
- OpenDate: Der Zeitpunkt des Anlegens der Version
- ReleaseDate: Der Zeitpunkt der Freigabe
- ExecTime: Die tatsächliche Bearbeitungszeit

- RespPers: Die verantwortliche Person
- #OfPers: Die Anzahl der mitwirkenden Personen
- Operat: Der Typ von Operation, die zu dieser Version geführt hat (siehe Kapitel 1.2)
- Phase: Die Phase, in der die Version entstanden ist (Kapitel 1.3)
- UsedDocs: Die Dokumente, die bei der Erstellung der Version benutzt wurden.
- Comment: Ein Kommentar

2.2 ProjectDictionary-Template

Um unter anderem eine einheitliche Namengebung zu gewährleisten, werden global sichtbare Namen, Datentypen usw. in ein gemeinsames Dictionary geschrieben. Dies dient als Referenz bei der Benennung neuer Objekte, Datentypen, Funktionen und Signale.

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*

Terms

ModelName	*FullName*	*Description*	*GermanName*

Datatypes

ModelName	*FullName*	*Range*	*Description*

Functions

ModelName	*FullName*	*Parameters*	*Description*

Signals

ModelName	*FullName*	*Parameters*	*Description*

ChangeRequests

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Abbildung 5: ProjectDictionary-Template

Die Kopfzeile enthält dokumentspezifische Angaben:

- DocName: Der Name des Dokuments, entspricht dem *Modelname*
- Project: Der Projektname
- Version: Die aktuelle Versionsnummer
- FullName: Der volle Name des Dokuments
- ShortDescription: Ein Kommentar zum Dictionary

Im weiteren wird auf diesen teil der Dokumente nicht weiter eingegangen.

In der nächsten Tabelle werden die Begriffe, die innerhalb des Projekts auftreten, erklärt:

- ModelName: Der Name, wie er innerhalb der Modellierung benutzt wird
- FullName: Der ausführliche Name
- Description: Eine Erläuterung des Eintrags
- GermanName: Die deutsche Übersetzung

Desweiteren werden im Dictionary die Datentypen global definiert:

- ModelName: Der Name bei der Verwendung
- FullName: Der ausführliche Name
- Range: Bewegen sich zulässige Werte des Datentyps innerhalb eines Bereichs, wird dieser Bereich hier definiert.
- Description: Eine textuelle Darstellung des Datentyps. Handelt es sich um einen komplexeren Datentyp, so muß das Aussehen hier definiert werden. Des kann mit einer C-ähnlichen Notation geschehen.

Die globalen Funktionen werden wie folgt definiert:

- ModelName: Der Name bei der Verwendung
- FullName: Der ausführliche Name
- Parameters: Die Typen der Parameter, getrennt durch Kommata.
- Description: Eine Erklärung der Funktion.

Die Signale werden mit Hilfe folgender Attribute beschrieben:

- ModelName: Der Name bei der Verwendung
- FullName: Der ausführliche Name
- Parameters: Die Typen der Parameter, getrennt durch Kommata.
- Description: Eine Erklärung des Signals.

Die ChangeRequest-Einträge dokumentieren notwendige Änderungen, die nicht von der jeweils verantwortlichen Person gefunden wurden. Sie resultieren aus Einträgen in einer globalen Liste (siehe auch Kapitel 2.5 und Kapitel 3.5). und werden durch folgende Einträge beschrieben:

- RequestNo: Die Nummer aus der globalen Liste
- Type: Der Typ des Eintrags: *Error*, *Fault* oder *Change*
- Method: Die Operation, in der der Fehler/Fault/Change entdeckt wurde. Dies kann z.B. *review* oder *test* während einer Verifikationsoperation des jeweiligen Dokuments gewesen sein. Wenn er während einer Operation auf einem anderen Dokument gefunden wurden, so trägt man hier die entsprechende Operation mit einem vorgestellten *ext* für *external* ein.
- Tester: Der Name des Entdeckers
- Description: Die eigentliche Fehlerbeschreibung. Diese kann auch schon einen Lösungsvorschlag beinhalten.
- Transferred: Der Zeitpunkt der Übertragung aus der globalen *ChangeRequest*-Liste
- Corrector: Der Name des Entwerfers, der den Fehler behoben hat.
- Corrected: Der Zeitpunkt der Korrektur. Ein Eintrag an dieser Stelle ist gleichbedeutend damit, daß der Fehler behoben wurde.

Auch dieser Teil der Beschreibung taucht in allen weiteren Dokumenten auf und wird da nicht mehr neu beschrieben!

In Abbildung 6 ist wiederum ein Beispiel dargestellt.

ProjectDictionary

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*
prjDict	BaX2	v1	projectDictionaryBaX2	consists terms, global datatypes and so on as a reference for all further definitions

Terms

ModelName	*FullName*	*Description*	*GermanName*
actUser	actual user	current user of an area	aktueller Benutzer
actVal	actual Value	current setting of a variable	Istwert

Datatypes

ModelName	*FullName*	*Range*	*Description*
Celsius	Celsius	[-100,100]	unit of a temperature

Functions

ModelName	*FullName*	*Parameters*	*Description*
printBi	print Binary	String, Binary, Time	prints the string and the binary value with the given time to the stnadard output

Signals

ModelName	*FullName*	*Parameters*	*Description*
getMotDet	get motion Detector		requests the actual value of a motion detector.

ChangeRequests

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Abbildung 6: Ein Beispiel eines ProjectDicitonaries

2.3 ProblemDescription-Template

Die ProblemDescription bildet eines der Dokumente, die als Eingabe in den gesamten Analyseprozeß dienen. Hier wird in informeller Weise das BlackBox-Verhalten des Systems aus Be-

ProblemDescription

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*

Description

Needs

ModelName	*Description*

ChangeRequests

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Abbildung 7: ProblemDescription-Template

nutzersicht beschrieben. In der Beschreibung auftauchende, allgemeingültige Begriffe werden in dem PrjDict definiert.

Die Beschreibung gliedert sich in vier Teile. Im ersten Teil werden wieder dokumentspezifische

Informationen abgelegt (siehe Kapitel 2.2).

Anschließend erfolgt ein allgemeiner Beschreibungsteil. Hier können z.B. Annahmen oder allgemeine Festlegungen für die weitere Beschreibung gemacht werden.

Im dritten Teil der ProblemDescription wird das Systemverhalten in Form von Needs beschrieben.

- Modelname: Die Needs werden in verschiedene Typen aufgeteilt (z.B. User-, Facility-Manager- oder Ownerneeds) und jeweils durchnumeriert. Diese Numerierung ergibt den Modelname.
- Needs: Hier erfolgt die eigentliche Beschreibung der Needs. Diese erfolgt primär in informeller Weise, kann jedoch auch andere, formalere Techniken beinhalten.

Der letzte Teil ist wieder in Kapitel 2.2 beschrieben.

2.4 BuildingDescription-Template

Für die Beschreibung eines Gebäudes mit seiner Installation wird ein sehr abstraktes Template vorgegeben, da man sich hier viele Beschreibungsformen vorstellen kann. Neben einer informellen Art bietet es sich auch an, das Gebäude mithilfe eines CAD-Programms zu beschreiben. Um die Modellierer hier nicht von vornherein einzuschränken, sind hier keine Vorgaben gemacht. Dementsprechend sieht das Template folgendermaßen aus:

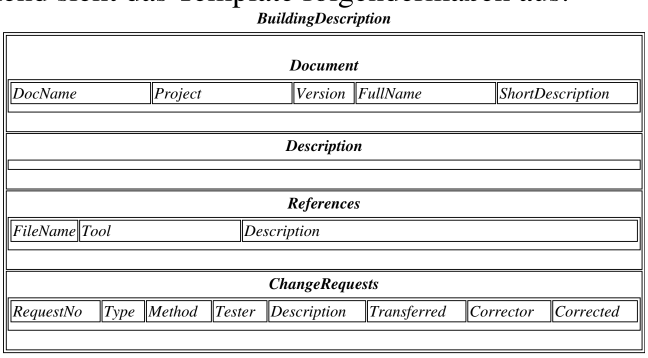

Abbildung 8: BuildingDescription-Template

Nach dem ersten Teil kann die informelle Beschreibung abgelegt werden:

- Description: Die informelle Beschreibung des Gebäudes und seiner Installation

Sollten zur Beschreibung weitere, eventuell mit externen Tools erstellte Dateien benutzt werden, werden sie in der folgenden Tabelle aufgelistet:

- FileName: Der Name der Datei
- Tool: Das Werkzeug, mit dem die Datei bearbeitet werden kann
- Description: Kurze Beschreibung des Inhalts

Aufgrund der Länge eines solchen Dokuments wird hier auf ein Beispiel verzichtet.

2.5 ChangeRequest-Template

Alle Changes, Fehlverhalten oder Fehler werden bei Auffinden in eine globale Liste, die ChangeRequest-Liste eingetragen.

Errors/Changes

No error/ change	*Date*	*Time*	*Type*	*Phase*	*Op*	*Document Reason*	*Tester*	*Description*	*Documents affected*	*Date*	*Time*	*Correctors*

Abbildung 9: Template für die ChangeRequest-Liste

Dabei bedeuten die Einträge im einzelnen:

- RequestNo: Die global eindeutige Request-ID
- Date: Das Datum der Entdeckung
- Time: Die Uhrzeit der Entdeckung
- Type: Die Art des Eintrags: *Error*, *Fault* oder *Change*
- Phase: Eine der in Kapitel 1.3 vorgestellten Phasen, in der sich das Dokument, das bei *DocumentReason* eingetragen wird, befindet.
- Op: Die Operation, in der der Fehler/Fault/Change entdeckt wurde. Dies kann z.B. *review* oder *test* während einer Verifikationsoperation des jeweiligen Dokuments gewesen sein. Wenn er während einer Operation auf einem anderen Dokument gefunden wurden, so trägt man hier die entsprechende Operation mit einem vorgestellten *ext* für *external* ein.
- DocReason: Das Dokument, das bei der Entdeckung des Fehlers bearbeitet wurde.
- Tester: Der Name des Entdeckers
- Description: Die eigentliche Fehlerbeschreibung. Diese kann auch schon einen Lö-

sungsvorschlag beinhalten.

- Docs aff.: Die Namen der Dokumente, die betroffen sind (falls bekannt).
- Date: Das Datum, an dem der Eintrag in das entsprechende Dokument übernommen wurde.
- Time: Die Uhrzeit, an dem der Eintrag in das entsprechende Dokument übernommen wurde.
- Correctors: Die Namen der Entwickler, die den Eintrag gelesen und bei Bedarf bei den entsprechenden Dokumenten übernommen haben.

Ein Beispiel kann folgender Abbildung entnommen werden.

Errors/Changes

No error/ change	*Date*	*Time*	*Type*	*Phase*	*Op*	*Document Reason*	*Tester*	*Description*	*Documents affected*	*Date*	*Time*	*Correctors*
1	12.1.99	12:24	Error	ReqMod	ref	Door	Queins	MotDet returns no "true" if entering a door	MotDet	12.1.99	12:56	Kronenburg
2	12.1.99	12:27	Change	reqMod	ext	LightedArea	Peper	Door should send a signal *newEntrance* if entering a room	Door	12.1.99	13:02	Queins

Abbildung 10: Beispiel einer ChangeRequest-Liste

2.6 ObjectStructure

Dieses zentrale Dokument beschreibt den Systemaufbau. Hier kommen alle im System verwandten Objekttypen mit ihren Beziehungen untereinander vor.

ObjectStructure

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*

ObjectTypes

ModelName	*InputSignalList*	*OutputSignalList*	*Components*	*Number*	*Type*

ChangeRequests

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Abbildung 11: Template für die Objektstruktur

Die Objekttypen werden im einzelnen durch folgende Eigenschaften charakterisiert:

- ModelName: Der Name des Objekttyps, wie er im Modell verwandt wird.
- InputSignalList: Die eingehenden Signale.
- OutputSignalList: Die Signale, die vom Objekttyp versandt werden.

- Components: Die Namen der aggregierten Komponenten.
- Number: Die Anzahl der Komponenten eines Typs.
- Type: Der Objekttyp der Komponenten.

Bei der Definition entwickelt sich eine Tabelle, bei der in der ersten Zeile eines Objekttyps die ersten drei Spalten, und in allen weiteren Zeilen die letzten drei Spalten ausgefüllt werden.

ObjectStructure

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*
objStr	Bax2	v1	objectStructureBaX2	contains a descriptoion of the object hierarchy, designed for the control of heating and lighting of the 4th floor of building 32.

ObjectTypes

ModelName	*InputSignalList*	*OutputSignalList*	*Components*	*Number*	*Type*
Door	DoorInp=(getDoor)	DoorOutp=(newDoor, newEntrance)			
			imd	2	MotDet
			dcc	1	Switch
MotDet	MDInp=(getMD)	MDOutp=(newMD)			

ChangeRequests

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Abbildung 12: Beispiel einer Objektstruktur

2.7 Tasks

Dieses Dokument dient dazu, die Gedanken, die in der ersten Phase zur Erstellung der Objektstruktur führten, festzuhalten. Viele Objekttypen resultieren aus der Zerlegung bzw. Zuweisung von einzelnen Aufgaben (*Tasks*), die hier beschrieben werden sollen.

Tasks

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*

Tasks

ModelName	*FullName*	*Description*	*SubTasks*	*AssocNeeds/Tasks*	*ObjectType*

ChangeRequests

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Abbildung 13: Task-Template

Bei einer Task werden folgende Attribute beschrieben:

- ModelName: Der Name des Task. Sollte nicht zu lang sein und dient nur, um diesen Task bei der Definition anderer Tasks referenzieren zu können. Als einfachste

Variante bietet sich hier eine einfache Numerierung an.

- FullName: Ein aussagekräftiger Name für die Task.
- Description: Die Beschreibung der Tasks.
- SubTasks: EIn Verweis auf eine Task, die aus der aktuellen Task folgt.
- AssocNeeds/
  Tasks: Eine Liste der Needs bzw. Tasks, aufgrund derer die jeweilige Task benötigt wird.
- ObjectType: Der Objekttyp, dem die Task zugeordnet werden soll.

Ein kleines Beispiel ist in Abbildung 14 dargestellt.

Tasks

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*
tasks	BaX2	v1	tasksBaX2	contains the main tasks, defined in the first phase of the analysis process

Tasks

ModelName	*FullName*	*Description*	*SubTasks*	*AssocNeeds/Tasks*	*ObjectType*
T1	DoorOpening	if a door opens, propagate a possible new entrance	T2		Door
T2	SwitchChange	report the switch value at every change on it		T1	Switch

ChangeRequests

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Abbildung 14: Beispiel für die Taskliste

2.8 ObjectType-Template

Wie schon erwähnt, wird in der ersten Phase das zu beschreibende System in Objekttypen zerlegt. Die einzelnen Objekttypen werden in der zweiten Phase durch das in Abbildung 15 dargestellte Template beschrieben werden.

ObjectType

Document

DocName	Project	Version	FullName	ShortDescription

Description

Components

ModelName	Number	Type	Version

Tasks

ModelName	Description	Strategy	AssocNeeds/Tasks

Attributes

ModelName	FullName	Type	Value	Description

Signals

ModelName	FullName	Type	Components	Parameters	Description

ChangeRequests

RequestNo	Type	Method	Tester	Description	Transferred	Corrector	Corrected

Abbildung 15: Template für einen Objekttyp

Die Beschreibung eines einzelnen Objekttyps gliedert sich wie folgt:

- Description: Eine informelle Beschreibung, die kurz die Funktionalität darstellt.

Der Aufbau der Aggregationshierarchie wird in dem folgenden Abschnitt beschrieben.

- ModelName: Die Namen der aggregierten Komponenten eines Typs. Die Namen der unterschiedlichen Komponenten werden eindeutig über eine Nummer (siehe nächster Eintrag), die an den Namen angehangen wird. Dementsprechend sollte der ModelName kein numerisches Zeichen enthalten. Weiterhin sollte er nicht zu lang sein (ca. 3 Buchstaben) und leitet sich i.A. aus dem Namen des jeweiligen Typs ab.
- Number: Die Anzahl der Komponenten des jeweiligen Typs. Aus dem ModelName und einer aufsteigenden Zahl ergibt sich der Name einer Komponente, wie er in der weiteren Entwicklung benutzt wird.
- Type: Der Typ der Komponenten.
- Version number: Handelt es sich bei der aktuellen Version des Objekttyps um eine Produktkonfiguration, so wird hier die Version des entsprechenden Typs der jeweiligen Komponenten eingetragen. Diese Version beschreibt i.A. wiederum

eine Produktkonfiguration. In den meisten Fällen wird diese Version die aktuellste Version sein.

Nach der Definition des Aufbaus des Systems wird eine informelle Beschreibung dessen gegeben, was der Objekttyp leisten soll. Diese Beschreibung findet im Abschnitt *Tasks* statt. Die Task werden entweder neu definiert oder werden detaillierter beschrieben, wenn sie schon in der globalen Taskliste (siehe Kapitel 2.7) definiert wurden.

- ModelName: Der Name des Task. Sollte nicht zu lang sein und dient nur, um diesen Task bei der Definition anderer Tasks referenzieren zu können. Als einfachste Variante bietet sich hier eine einfache Numerierung an.
- Description: Die eigentliche Aufgabenbeschreibung. Falls die Modellierer bereits Erfahrung in der Umsetzung der Tasks in ein SDL-Modell besitzen, kann eine ungefähre Komplexität als Richtlinie angegeben werden: Die Tasks sollten so einfach sein, daß sie möglichst auf eine Seite in den SDL-Modellen paßt.
- Strategy: Erste Realisierungsvorschläge. Die Form, in der diese beschrieben werden, ist frei wählbar. Neben der rein sprachlichen kann man sich hier auch kleine Automaten oder Tabellen denken.
- AssocNeeds/ Tasks: Eine Liste der Needs bzw. Tasks, aufgrund derer die jeweilige Task benötigt wird.

Aus der Beschreibung der Funktionalität, insbesondere der Strategien, können auch einige Attribute des Objekttyps folgen, die durch folgende Einträge beschreiben werden.

- ModelName: Der Name, wie er in der Modellierung verwandt wird
- FullName: Der volle, aussagekräftige Name
- Type: Der zugehörige Datentyp
- Value: Ein initialer Wert des Attributes. Handelt es sich bei dem Attribut um eine Konstante, ändert sich dieser natürlich nicht über die Lebensdauer des Systems.
- Description: Eine kurze Beschreibung

Die Schnittstelle des Objekttyps nach außen, d.h. in der Aggregationshierarchie nach oben, und die benötigte Kommunikation mit den aggregierten Komponenten, wird im folgenden Abschnitt beschrieben.

- Modelname: Der Name, wie er in der Modellierung verwandt wird
- Fullname: Der Name des Signals
- Type: Der Typ des Signals. Man unterscheidet die Typen *INDEF* (Signal kommt in den Objekttyp, wird bei diesem Objekttyp definiert), *INREF* (Signal kommt in den Objekttyp, ist bei einem anderen Objekttyp definiert) und die entsprechenden Typen *OUTDEF* und *OUTREF.* Weiterhin exisitieren die Typen *UP* und *DOWN*, um Signale zu kennzeichnen, die nur durch den entsprechenden Objekttyp nach oben bzw. unten durchgereicht werden.
- Components: Sie geben an, zu welchen Komponenten das jeweilige Signal gesandt wird, bzw. woher es kommt. Hier können auch mehrere Einträge vorhanden sein. Kein Eintrag bedeutet Kommunikation mit dem in der Hierarchie oberhalb liegenden Objekttyp.
- Parameters: Die Typen der Parameter in der richtigen Reihenfolge des Signals.
- Description: Eine kurze Beschreibung.

Die folgende Abbildung zeigt ein unvollständiges Beispiel eines Objekttyps:

ChangeRequests

ObjectType

RequestNo	*Type*	*Method*	*Tester*	*Description*	*Transferred*	*Corrector*	*Corrected*

Document

DocName	*Project*	*Version*	*FullName*	*ShortDescription*
LightedArea	BaX2	v1	LightedAreaBaX2	

Description

This object controls the light of an area (room, hallway etc.). This area distinguishes two states: occupied (occ) and not occupied (empty).

Components

ModelName	*Number*	*Type*	*Version*
cl	2	Light	

Tasks

ModelName	*Description*	*Strategy*	*AssocNeeds/ Tasks*
T1	Determine a lightscene (without setting the lights)	if the user chooses a lightscene, this one will become the actual lightscene (als) which has to be used if the room is reoccupied within the light scene reset time (lrt), reestablish the last state of the lights, otherwise set a default light scene (dls) as the actual light. If the room is unoccupied use the empty light scene (els)	

Attributes

ModelName	*FullName*	*Type*	*Value*	*Description*
als	actual light scene	LightSceneType	('EMPTY',0,0,(1,2,3)	current desired light scene
lrt	light scene reset time	Duration	60	time to reset the light scene

Signals

ModelName	*FullName*	*Type*	*Components*	*Parameters*	*Description*
setLSReeTime	setLightSceneResetTime	InDef		Duration	

Abbildung 16: Ein Beispiel eines Objekttyps

3 Durchführung der Analyse

In diesem Abschnitt wird die Durchführung einer Analyse betrachtet. Dazu werden die einzelnen Schritte genauer mit jeweils einigen Modellierungsrichtlinien vorgestellt. Für den SDT-Entwurf wird auf Kapitel 4 verwiesen.

3.1 Entwurf der Objektstruktur

Aufgabe dieses Schritts ist die Erstellung einer Aggregationshierarchie, in der alle im System vorkommenden Objekttypen eingegliedert sind. Die entstehende Typ-Instanzen-Hierarchie kann natürlich nur eine erste Version sein, zusätzlich benötigte Objekttypen können im späteren Verlauf noch eingefügt werden.

Ausgangspunkt ist i.A. ein Modell des Gebäudes, auf das sich die Problembeschreibung bezieht. Durch diese strukturelle Aggregationshierarchie ist dann ein Grundgerüst gegeben. Dieses muß nun noch um Objekttypen erweitert werden, die die Funktionalität tragen, die nicht sinnvoll einem Objekttyp des Gebäudes zugeordnet werden kann. Dies können komplexe Controller, Sensuatoren oder auch Objekttypen sein, die einen Effekt (Licht, Heizung, ...) in seiner Gesamtheit betrachten. Als Grundlage hierfür kann die in Team2 entwickelte Referenzarchitektur dienen.

Bei der Erstellung ist darauf zu achten, daß ein Objekttyp aus nicht übermäßig vielen Komponenten besteht. Eine Obergrenze sollte hier 8 Objekte sein. Der Grund liegt in der Darstellbarkeit innerhalb des Werkzeugs SDT begründet.

Gleichzeitig mit der Erstellung der Hierarchie können erste, grobe Funktionen in die zur Verfügung gestellten Templates eingetragen werden. Sie bilden eine Art von Plausibilität für den jeweiligen Objekttyp. Gleichzeitig werden dadurch die Gedanken, die zu diesem Objekttyp führten, dokumentiert.

Natürlich kann bei diesem Entwurf auch auf die Erfahrung aus früheren Entwürfen zurückgegriffen werden, insbesondere dann, wenn schon einmal ein Gebäude gleichen Typs betrachtet wurde.

Die Hierarchie wird zunächst dadurch gebildet, daß in HTML die entsprechenden Objekttypen mit dem geeigneten Template (Kapitel 2.6) angelegt werden. Dort werden dann die aggregierten Komponenten mit ihren Typen eingetragen.

3.2 Entwurf der RequirementDescription

In dieser Phase werden die Beschreibungen der einzelnen Objekttypen weiter verfeinert. Erste Aufgabe ist es, die einzelnen Tasks mit ihren jeweiligen Strategien herauszufinden und zu beschreiben. Dabei sollte darauf geachtet werden, daß diese Tasks in ihrem Umfang nicht zu komplex werden. Von Vorteil ist es, wenn die einzelnen Tasks getrennt voneinander zu modellieren sind. Es sollen nicht allzu viele Berührungspunkte zwischen einzelnen Tasks existieren. Die Strategien geben eine erste Realisierungsmöglichkeit der Tasks an. Dabei kann es schon vorkommen, daß sich Gründe für die Existenz von Attributen ergeben. Diese Attribute werden dann beim Objekttyp definiert.

Die Signale der einzelnen Objekttypen geben die Schnittstelle an. Sie resultieren aus Anforderungen, die an den Objekttyp gestellt werden. Sie müssen ebenfalls in dieser Phase mit ihren Parametern definiert werden.

Beim Entwurf der Tasks ist darauf zu achten, daß alle Needs, die dem Objekttyp zugeordnet sind, berücksichtigt wurden. Desweiteren ergeben sich die Aufgaben durch Anforderungen von anderen Objekttypen. Diese werden durch den Change-Mechanismus (siehe auch Kapitel 1.2.3) dem betroffenen Objekttyp bekannt gemacht und können dann als Task realisiert werden.

Entscheidungen in dieser Phase werden in die Objekttyp-spezifischen Dokumente eingetragen (Kapitel 2.8). Weiterhin muß das in der ersten Phase entstandene Dokument *ObjStr* konsistent weitergeführt werden.

3.3 Verifikation der RequirementDescription

Dieser Verifikationsschritt wird in drei Teile unterteilt, die prinzipiell parallel zueinander durchgeführt werden können: eine objektbezogene Verifikation, eine objektübergreifende Verifikation und eine Verifikation, die z.B. die Kommunikation eines Objekttyps mit seinen aggregierten Objekten betrachtet.

Die objektbezogene Verifikation wird durch Inspektion des jeweiligen Objekttyps durchgeführt und erhält als Eingabe die Problembeschreibung (daraus insbesondere die Needs) und eine Checkliste mit einer Reihe von Punkten, die für jedes Objekt überprüft werden müssen.

Die objektübergreifende Verifikation dient dazu, die Zerlegung der Needs zu überprüfen. Dazu werden alle Tasks aller Objekte zusammengetragen und den Needs entsprechend geordnet.

Nun wird für jeden einzelnen Need überprüft, ob die ihm zugeordneten Tasks in ihrer Gesamtheit den Need realisieren.

Um die Kommunikation zu checken, wird ein Konsistenzprüfung bezüglich der ausgetauschten Signal zwischen den beteiligen Objekttypen vorgenommen. Das bedeutet, es muß überprüft werden, ob die in dem aktuellen Objekttyp erwarteten bzw. ausgesandten Signale mit denen der entsprechenden Komponenten übereinstimmen.

3.4 Verifikation der SDL-Objekte

3.5 Fehler/Changes-Behandlung

Während der Analyse wird es immer wieder vorkommen, daß Fehlverhalten oder Fehler gefunden bzw. Änderungen notwendig werden. Hier muß man prinzipiell unterscheiden, ob die Fehler/Faults/Changes (im folgenden Defect) bereits einem Dokument oder einem verantwortlichen Entwerfer zugeordnet werden können oder nicht. Auf jeden Fall muß der Defect protokolliert werden. Dies geschieht durch einen Eintrag in die globale ChangeRequest-Liste. Bei dieser Eintragung wird eine eindeutige Fehler-ID vergeben. Falls der verantwortliche Entwerfer bekannt ist, kann diesem (z.B. über E-Mail) diese Fehler-ID mitgeteilt werden, und er kann dann den Zeitpunkt bestimmen, wann er den Eintrag in das entsprechende Dokument übernimmt und wann er den Defect behebt. Ist keine verantwortliche Person bekannt, oder läßt sich der Defect nicht eindeutig einem Dokument zuordnen, wird die Fehler-ID per Broadcast an alle Entwickler bekanntgegeben. Diese müssen dann entscheiden, wie sich der Fehler aufteilt.

Das Kopieren des Defect-Eintrags aus der globalen Defect-Liste in das betroffene Dokument wird notwendig, da sich ein Defect auf mehrere Dokumente auswirken kann und so der Defect eventuell aufgespalten und dementsprechend verändert werden muß. Jedoch ist die Verfolgbarkeit durch Angabe der eindeutigen Fehler-ID gewährleistet.

3.6 Erstellung der TestCases

3.7 Prototyp-Test

Der Prototyping-Schritt dient dazu, das Verhalten des Systems oder Teile von ihm zu untersuchen. Nach Erzeugen eines Prototyps (siehe Kapitel 4.3) werden die in Kapitel 3.6 definierten TestCases durchlaufen und eventuelles Fehlverhalten dokumentiert. Die nächste Aufgabe ist es, die Fehlerursachen zu lokalisieren und sie zu beheben. Dazu werden die Fehler in die Errorliste eingetragen und durch eine correct-Operation bearbeitet.

In den wenigsten Fällen macht es Sinn, ein Kontrollsystem stand-alone, d.h. ohne eine Umgebung, zu testen. Die Umgebung kann entweder durch ein real existierendes Gebäude oder durch einen Simulator gegeben sein. Es ergibt sich folgender Testaufbau:

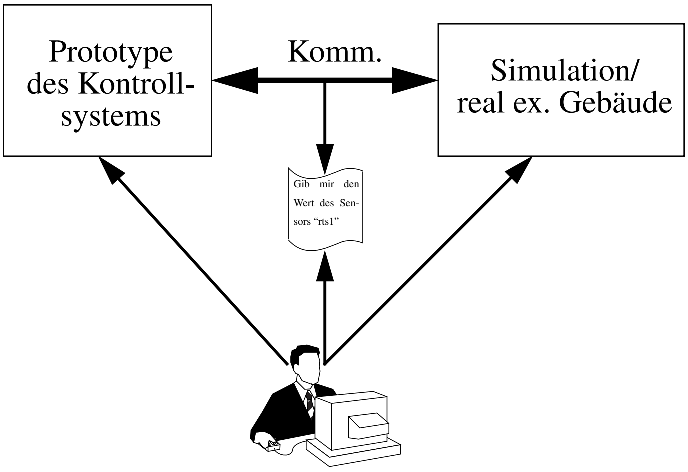

Abbildung 17: Testaufbau zum Prototyping

Der Entwickler hat die Möglichkeit, das Kontrollsystem oder den Simulator zu beeinflussen. Dadurch wird es ermöglicht, z.B. bestimmte, in den TestCases beschriebene Szenarien zu erstellen. Beobachtet werden Reaktionen am Simulator, Ausgaben des Kontrollsystems (spezifizierte oder nur zu Debugzwecken erzeugte) und ein Trace der Kommunikation zwischen Kontrollsystem und Simulator.

4 Richtlinien für den SDL-Entwurf

Beim Entwurf eines SDL-Dokuments sollten einige Richtlinien eingehalten werden, die mehrere Ziele unterstützen:

- Zeitersparnis beim Erstellen des Modells
- Verständlichkeit und Lesbarkeit des Modells
- Wiederverwendung von einzelnen Ausschnitten
- Simulation und Prototyping des Modells
- Anbindung an einen Gebäudesimulator oder den Testraum

Zunächst werden alle zu benutzenden SDL-Entities (Blocktypen, Instanzenblöcke, Prozesse, Automaten, Sensoren, Aktuatoren, System, Datentypen, Funktionen, Signale) kurz vorgestellt (Kapitel 4.1). Für sie existieren SDL-Templates, die leicht an die gewünschte Modellierung angepaßt werden können (Kapitel 4.2). Durch die Verwendung der Templates wird ein nicht unerheblicher Teil des Aufwands zur Erstellung der Modelle eingespart. Den Abschluß bildet ein kurze Einführung in das Generieren von Prototypen (Kapitel 4.3).

4.1 Beschreibung der vorkommenden SDL-Strukturen

4.1.1 Blocktypen

Bei der Beschreibung eines Systems werden alle vorkommenden Objekttypen in einer baumartigen Struktur zusammengefaßt, die eine Aggregationshierarchie beschreibt. Ein Objekttyp in diesem Baum wird auf einen Blocktyp in SDL abgebildet, der aus einem Controllerblock (einem Prozeß oder mehreren Services) und aus mehreren Komponenten besteht. Neben einer eventuellen Weiterleitung der Signale von bzw. zu den Komponenten nach bzw. von außen wird auch die Funktionalität implementiert, die dem jeweiligen Blocktyp direkt zugeordnet ist.

4.1.2 Panel/Monitor

Neben den Komponenten und einem Controllerblock zur Realisierung der gewünschten Funktionalität existieren noch die beiden Blöcke *Panel* und *Monitor*. Durch sie wird es ermöglicht, bestimmte Funktionalität an definierten Stellen zu beschrieben. Dies bezieht sich auf Eigenschaften der Systemkomponenten, die typisch für ein Steuerungssystem sind. Es sind dies z.B. die Möglichkeit, manuelle Eingaben zu tätigen oder Werte anzuzeigen (Panel) oder auch Verläufe mit zu protokollieren (Monitor). Die Funktionalität des Panels wird im allgemeinen in der späteren Realisierung von HW-Komponenten übernommen. Da diese aber im Analysedokument nicht zur Verfügung stehen, wird deren Funktionalität in einer eigenen Komponente beschrieben.

4.1.3 Instanzenblöcke

Um eine korrekte Instantiierung von Blocktypen durchzuführen ist es notwendig, sowohl Werte von Attributen als auch Signalnamen instanzenspezifisch zu vergeben. Da dies in der jetzigen Version des SDL-Tools SDT nicht möglich ist, wird für jede Instanz ein Block (Instanzenblock) eingeführt, in dem das instanzenspezifische modelliert wird. Dieser Block wird zwischen die Instanz und den Controller-Block des jeweiligen Blocktyps plaziert, sodaß der Controller z.B. anhand des Signalnamens entscheiden kann, welche Instanz ein Signal gesandt hat

Die Namen der Signale zwischen Controllerblock und Instanzenblock werden durch eine Konkatenation des original Signalnamens und des Instanzennamens gebildet. In dem Controller kann dann entschieden werden, ob bei Bedarf das Signal mit oder ohne dem Instanzennamen nach außerhalb weiter gereicht wird. Die Definition der instanzenspezifischen Signale geschieht innerhalb der Blocktypdefinition, in der sie referenziert werden.

Eine andere Möglichkeit zur Unterscheidung der Instanzen im Controllerblock besteht im Setzen eines entsprechenden Parameters in dem Instanzenblock. Dieser Parameter kann dann im Controllerprozess ausgewertet werden. Diese Möglichkeit bietet sich bei solchen Signalen an, die von vornherein einen solchen Parameter besitzen. Ist eine Unterscheidung nicht notwendig, z.B. wenn das Signal nur nach außen weitergereicht werden muß, kann dieser Parameter entfallen.

Durch die Einführung der Instanzenblöcke wird es erst ermöglicht, das vollständige System zu simulieren oder einen Prototyp zu generieren.

4.1.4 Sensoren/Aktuatoren

Die Sensoren und Aktuatoren werden in Form einer Bibliothek (eines SDL-Packages) zur Verfügung gestellt. Vorlagen für die verschiedenen Typen von Sensoren/Aktuatoren sind als Templates gegeben.

4.1.5 SDL-System

Ein SDL-System besteht aus einer Instanz des Blocktyps, der den obersten Knoten in der gegeben ObjectStructure bildet. Diese Instanz hat Verbindungen zu einem Block *ProtoCtrl*, der alle im System vorkommenden Objekte verwaltet und die Verbindung des Systems mit der Außenwelt darstellt. Da diese Verbindung ausschließlich über Sensoren und Aktuatoren geschieht, besteht die Aufgabe der ProtoCtrl darin, zum einen die ankommenden Signale an die entsprechenden Objekte weiterzuleiten, zum anderen den physikalischen Sensuatoren Stellwerte bzw. Sensorabfragen zu übermitteln. Um diese Aufgabe erfüllen zu können, besitzt der ProtoCtrl Informationen über alle im SDL-System vorkommenden Objekte.

4.1.6 Datentypen, Funktionen und Signale

Datentypen, Funktionen und Signale werden global in einem eigenen Package definiert. Dies bedeutet, sie müssen einen systemweit eindeutigen Namen besitzen. Dies betrifft die Signale, die die Schnittstelle zwischen den Objekttypen bilden. Die instanzenspezifischen Signale werden innerhalb des jeweiligen Blocktyps definiert. Der Name der Signale kann dabei durch Verwendung des Namens (oder einer Abkürzung) des Blocktyps, in dem das Signal logisch definiert wird, eindeuitg gemacht werden. Die Funktionen sind zum Teil Funktionen, die der Kommunikation mit der Umgebung dienen, zum Teil Hilfsfunktionen, z.B. zu Debug-Ausgaben.

4.2 Verwendung der Templates

Hier soll nun beschrieben werden, wo in welchen Templates welche Änderungen durchgeführt werden müssen, um die Template an das gewünschte Modell anzupassen.

Als bereits definiert werden die Signale *dummy*, *ready* und *ident* angenommen. Das erste Signal dient ausschließlich dazu, die Templates syntaktisch korrekt formulieren zu können. Das ready-Signal zeigt an, daß die Initialisierung eines Objekts beendet ist. Dies ist dann der Fall, wenn alle seine Komponenten ein ready gesandt haben. Das ident-Signal schließlich dient dazu, den Namen der einzelnen Objekte dem ProtoCtrl bekannt zu machen.

4.2.1 Blocktypen

Ein Blocktyp wird normalerweise innerhalb eines SDL-Packages definiert. Seine Struktur mit seinen internen Signalen wird auf zwei SDT-Seiten beschrieben.

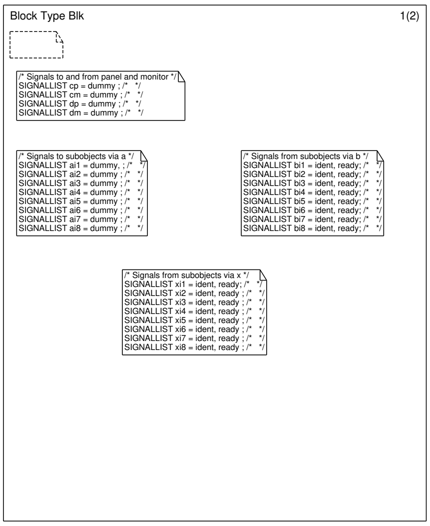

Abbildung 18: Signaldefinitionen

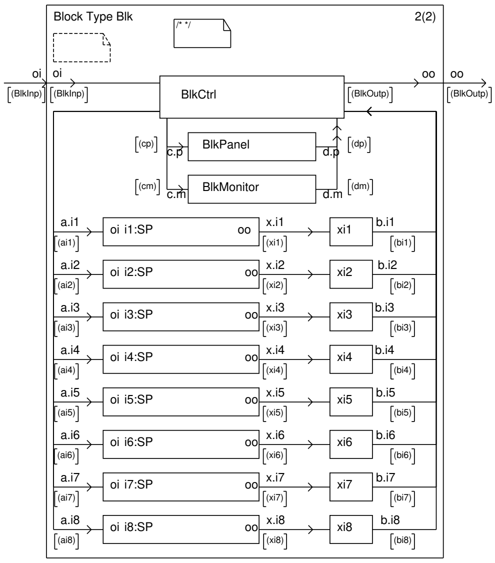

Abbildung 19: Strukturdefinition

Die Signallisten, die in den entsprechenden Blocktyp ein- bzw. austreten, werden direkt neben der Deklaration des Blocktyps definiert. Dies erleichtert das Wiederfinden der gewünschten Definitionen bei der Notwendigkeit einer Änderung.

Bei den Signaldefinitionen (Abbildung 18) werden die gegebenen Listen um die entsprechenden Signale erweitert. Dabei gelten für die Signale *bi<n>* die in Kapitel 4.1.3 angegebenen Richtlinien. Die Namen der Signallisten sollten nicht verändert werden. Für die Strukturdefinition müssen die Instanzen- und Typnamen ersetzt werden Der Name des Controllers mit seinen In/Out-Listen muß angepaßt werden. Nicht benötigte Instanzen werden mit ihren Instanzenblöcken gelöscht. Die Namen der Kanäle zu den aggegrierten Instanzen (*a.i**) können bei Bedarf ersetzt werden.Controller

Die Definition eines Controllers besteht aus zwei Dateien, die in Abbildung 20 und Abbildung

21 dargestellt sind.

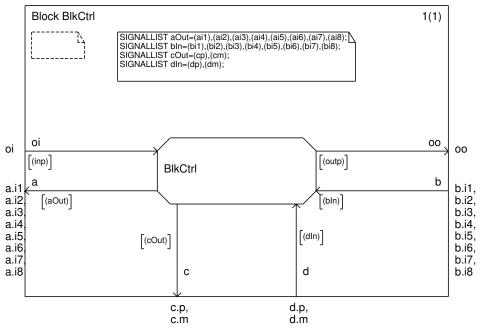

Abbildung 20: Blockdefinition

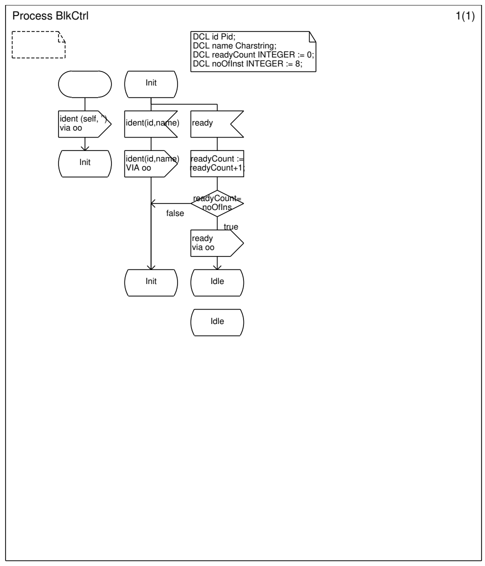

Abbildung 21: Prozeßdefinition

Bei der Blockdefinition werden die Ein/-Ausgabekanäle des Prozesses den Ports zugeordnet. Hier müssen neben dem Namen des Controllers die Signallisten *bIn* und *aOut* sowie die Anzahl der Ports *a.i** und *b.i** angepaßt werden. Eventuell muß zusätzlich (siehe Kapitel 4.2.1) eine Umbenennung der Ports *a.i** durchgeführt werden. Ebenfalls umbenannt werden die Ein/- Ausgabelisten *inp* und *outp* des Controllers.

in der zweiten Datei, der Definition der Prozesse mittels Zustandsgraphen muß eine Anpassung der Variablen *noOfInst* vorgenommen werden. Jedoch ist bei der Modellierung der Zustandsgraphen auf Übersichtlichkeit z.B. durch Aufsplitten der Graphen auf mehrere, logisch zusammenhängende Seiten zu achten.

4.2.2 Instanzenblöcke

Ähnlich wie die Controller werden die Instanzenblöcke in zwei Dateien beschrieben.

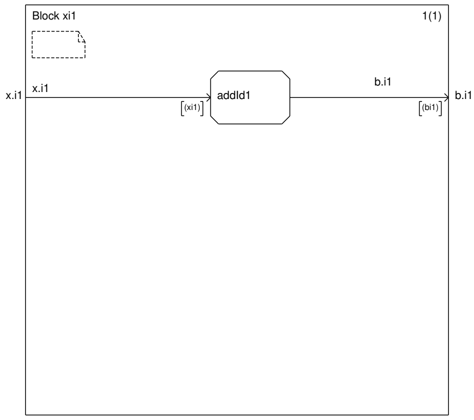

Abbildung 22: Instanzenblockdefinition

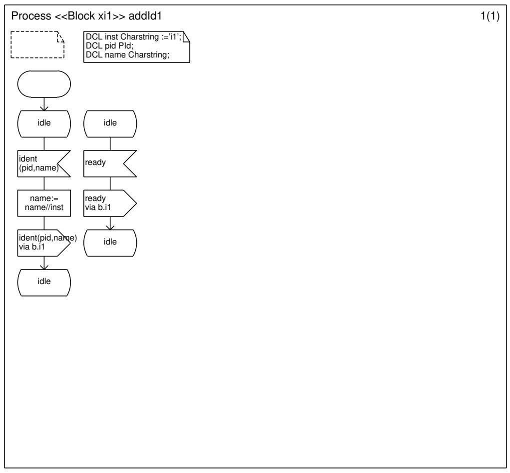

Abbildung 23: Prozeßdefinition

Bei der Blockdefinition müssen keine Änderungen vorgenommen werden.

In der Prozeßdefinition wird der Wert der Variablen *name* durch den Instanzennamen ersetzt und durch die gewünschten Signale erweitert.

4.2.3 ProtoControler

Die Schnittstelle des Systems mit der Umwelt geschieht durch einen sogenannten ProtoControler (siehe Kapitel 4.1.5). Für diesen ProtoControler existieren zwei Templates, die sowohl die Blockstruktur als auch den eigentlichen Prozeß vollständig definieren. An diesem Objekt müssen i.A. keine Änderungen durchgeführt werden.

4.3 Generieren von Prototypen

Durch das Generieren von Prototypen wird es ermöglicht, das System in seiner Gesamtheit zu testen. Jedoch ist es auch sinnvoll, einzelne Objekte oder Teile des Systems auf ihre Korrektheit zu überprüfen. Deswegen wird hier folgender Ansatz verfolgt:

Die gesamte Kontrollhierarchie wird in verschiedene Teile aufgeteilt, die getrennt voneinander entworfen werden können. Diese Aufteilung richtet sich nach der Baumstruktur, indem nur jeweils ein Blocktyp an oberster Stelle einer Partition existieren darf. Jede Partition wird in einem eigenen SDT-Package definiert, das andere Packages referenzieren kann. Der Name des Pakkage ergibt sich aus dem Namen des obersten Blocktypen. Das gesamte System kann dann zusammengesetzt werden, indem alle definierten Packages referenziert werden.

Neben dem Vorteil des parallelen Entwerfens bietet diese Art der Aufteilung zusätzlich noch die Möglichkeit, die Komponenten, also die einzelnen Packages, getrennt voneinander zu testen. Die wird dann möglich, wenn man in jedem Package zusätzlich mit Hilfe des ProtoControler, ein eigenes System definiert. Dadurch läßt sich dann ein System generieren, das nur den aktuellen Teilbaum der Kontrollhierarchie implementiert.

Zur Generierung der Prototypen muß folgendes existieren:

- Ein SDL-System, bestehend aus einem ProtoControler und einer Instanz der Wurzel des betrachteten Baums.
- der, bezogen auf die Hierarchie, nach unten vollständige Baum.
- für eine Simulation: eine Datei proc.c. Hier sind Hilfsfunktionen und die für die Kommunikation notwendige Anbindung über einen Socket an einen Simulator definiert.
- zusätzlich für eine Applikation: eine Datei app.c. In dieser Datei sind die von SDT vorausgesetzten Funktionen implementiert. Diese werden jedoch nicht ausgenutzt, um bei einer Applikation den gleichen Mechanismus zur Kommunikation des Kontrollsystems an die Aussenwelt wie bei einem Simulator zu gewährleisten.

Mit diesen Komponenten läßt sich in der durch SDT definierten Weise ein Prototyp erzeugen, der dann getestet werden kann.